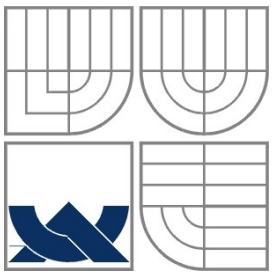


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

GENETICKÉ ALGORITMY A ROZVRHOVÁNÍ

GENETIC ALGORITHMS AND SCHEDULING

DIPLOMOVÁ PRÁCE
DIPLOMA THESIS

AUTOR PRÁCE — **BC. ONDŘEJ ŠKRABAL**
AUTHOR

VEDOUCÍ PRÁCE — **ING. JAN ROUPEC, PH.D.**
SUPERVISOR

BRNO 2010

ZADÁNÍ ZÁVĚREČNÉ PRÁCE

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav automatizace a informatiky
Akademický rok: 2009/10

ZADÁNÍ DIPLOMOVÉ PRÁCE

student(ka): Škrabal Ondřej, Bc.

který/která studuje v **magisterském studijním programu**

obor: **Aplikovaná informatika a řízení (3902T001)**

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Genetické algoritmy a rozvrhování

v anglickém jazyce:

Genetic Algorithms and Scheduling

Stručná charakteristika problematiky úkolu:

Problematika rozvrhování vede na úlohy s NP-složitostí. Z toho důvodu se jedná o typickou oblast pro nasazení heuristických optimalizačních metod, nasazení genetických algoritmů se zde jeví jako velmi vhodné vzhledem k jejich adaptabilitě a flexibilitě. Problémy činí zejména návrh vhodné modifikace genetického algoritmu pro daný problém, to bude také hlavním úkolem této práce.

Cíle diplomové práce:

Cílem práce je vybrat modifikace genetických algoritmů pro použití v rozvrhování strojírenské a plastikářské výroby. Vybrané algoritmy budou emplementovány v C++. Možnosti genetických algoritmů v dané oblasti budou porovnány s výsledky dalších heuristických metod.

Seznam odborné literatury:

Reeves, C. R. - Rowe, J. E.: Genetic Algorithms - Principles and Perspectives : A Guide to GA Theory, Kluwer Academic Publishers, 2003.
Brown, D. E. - Scherrer, W. T.: Inteligent scheduling systems, Kluwer Academic Publishers, 1995.
Prata, S.: Mistrovství v C++, Computer Press, Praha, 2007.

Vedoucí diplomové práce:Ing. Jan Roupec, Ph.D.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2009/10.

V Brně, dne 26.11.2009

L.S.

prof. RNDr. Ing. Miloš Šeda, Ph.D.
Ředitel ústavu

doc. RNDr. Miroslav Doupovec, CSc.
Děkan fakulty

ABSTRAKT

Práce se zabývá problémem rozvrhování výroby pro vstřikolisovnu plastových výrobků v konkrétním provozu. Řešení je založené na heuristických algoritmech, programovacích jazycích C+ + a C#, je postaveno na platformě .NET a LINQ do XML. Přináší srovnání základních heuristických metod s Genetickými algoritmy aplikovaných na problém v dané výrobě. Všechny metody a jejich výsledky jsou porovnány vzhledem k ručně sestaveným plánům.

ABSTRACT

This work deals with scheduling problem in particular plastic production service. The solution is based on heuristic algorithms, programming languages C + +, C # and is built on the .NET framework and LINQ to XML. It provides the users with comparisons of the heuristic approach with genetic algorithms applied to production problem. All methods results are compared in relation to hand-arranged plans.

KLÍČOVÁ SLOVA

rozvrhování, časová složitost, heuristické metody, Genetický algoritmus, C++, C#, generátor pseudonáhodných čísel, fitness funkce, .NET platforma, LINQ pro XML

KEYWORDS

scheduling, time complexity, heuristic methods, Genetic algorithm, C + +, C #, pseudorandom number generator, the fitness function, .NET framework, LINQ to XML

PODĚKOVÁNÍ

Touto cestou bych chtěl především poděkovat vedoucímu diplomové práce panu Ing. Janu Roupcovi, Ph.D. za věcné rady a připomínky při vypracování této diplomové práce. Dále pak panu Ing. Jaroslavu Hlavinkovi za ochotu při komunikaci s firmou Obzor, výrobní družstvo Zlín. V neposlední řadě bych poděkoval rodičům za finanční podporu a zázemí a také svému otci za pomoc při hledání firmy pro nasazení plánování.

Obsah:

1 ÚVOD

Tato práce vznikla na základě mého pobytu v Norsku na Univerzitě v Molde. Tato univerzita se zabývá logistikou a nabízí celou řadu zajímavých oborů. Práce je situována do této oblasti, přesněji se zabývá rozvrhováním a plánováním. Obecně problematika plánování řeší rozvržení úloh na zdroje tak, aby byl minimalizován celkový čas vytíženosti zdrojů a dosaženo maximálního zisku z provedených úloh. Často je nutné splnit další omezení, jako je čas ukončení nebo čas začátku úlohy, ale i jiné mnohem specifičtější podmínky. V této práci je možné seznámit se se základní problematikou rozvrhování. Problém rozvrhování je možné řešit pomocí celé řady algoritmů s využitím výpočetní techniky, proto je s tématem úzce spojena časová složitost algoritmu. Na základě této teorie je možné určit předpokládanou složitost algoritmu a odhadnout tak výpočetní čas potřebný k provedení tohoto algoritmu.

Práce nestuduje problém rozvrhování jen teoreticky, ale zabývá se konkrétním problémem ve výrobním družstvu Obzor Zlín, které bylo založeno v roce 1965. V současné době zaměstnává firma 390 pracovníků. Družstvo se zaměřuje na výrobu drobných elektronických výrobků, mechanických rozprašovačů a dávkovačů, kovových dílů, dílů pro automobilový průmysl, výsek těsnění a vstřikování plastů. [1]

Praktické problémy, tak jako i v jiných oblastech lidské činnosti, nevedou na triviální úlohy a často je složité nebo nemožné najít přesný matematický popis nebo algoritmus pro řešení dané úlohy. V některých případech úlohy sice přesný matematický popis mají, ale jejich exaktní řešení není možné se současnou výpočetní technikou dosáhnout v reálném čase. Z těchto důvodů byla vyvinuta celá řada heuristických metod, které často nenacházejí přesné řešení daného problému ve smyslu optimality. Tyto metody nezaručují nalezení globálního optima, takže nejúspornějšímu nebo finančně profitujícího rozvrhu se mohou vyhnout. Často se však tyto metody k řešením získaným na základě exaktních algoritmů značně přibližují.

Problém, kterým se práce zabývá, byl řešen právě na základě heuristických metod. Při hledání optimálního rozvrhu byly použity různé techniky a v praktické části je možné najít jejich srovnání. Práci dominují Genetické algoritmy jako jedna z heuristických metod. Tyto algoritmy se inspirují Darwinovou teorií evoluce.

Další neméně důležitou doménou práce je generování pseudonáhodných čísel s ohledem k řešenému problému. Při inicializaci výpočtu heuristické metody sestaví náhodně zvolený plán a snaží se ho vhodným způsobem modifikovat. Tyto modifikace často staví na náhodném postupu. Metody mohou uvíznout v lokálním minimu prohledávaného prostoru. Z těchto důvodů je podstatné mít kvalitní generátor pseudonáhodných čísel tak, abychom dokázali pokrýt co největší prohledávaný prostor řešení a byli schopni najít globální optimum, tedy exaktní řešení, nebo se mu značně přiblížit.

V rámci experimentální části práce byla v uvedené firmě provedena analýza problému, byl vybrán vhodný genetický algoritmus a byl implementován program pro sestavování rozvrhů. Samotná aplikace na sestavování rozvrhů výroby byla navržena v různých jazycích. Knihovna pro výpočty byla napsána v jazyce C++ v unmanaged kódu. Samotný grafický interface a operace s daty byly vytvořeny v jazyce C# pod platformou .NET framework. Tato část práce poskytuje dosažené výsledky na základě heuristických metod a jejich srovnání s předchozími zkušenostmi v daném výrobním závodu.

2 VYMEZENÍ ŘEŠENÉHO PROBLÉMU

Praktický úkol byl řešen ve firmě výrobní družstvo Obzor Zlín, kde vznikl požadavek na plánování výroby. Jak již bylo v úvodu řečeno, jedná se o větší firmu, která obsahuje několik oddělení a mimo jiné se skládá z oddělení na výrobu plastových dílů. Vstřikolisovna je vybavena stroji ENGEL a DEMAG. Další oddělení zajišťuje samostatné výrobní operace jako jsou vrtání, závitování, nýtování, apod. Další doménou jsou výroba a oprava vstřikovacích a vyfukovacích forem na CNC strojích. Poslední oblastí zájmu jsou kovo a elektro montáže. [1]

Rozvrhování výroby by mohlo být nasazeno do všech oddělení daného závodu, pro objem potřebného času k realizaci takového úkolu však bylo vybráno jen oddělení vstřikolisovny. Vstřikolisovna má 13 strojů, vyrábí 600 druhů výrobků z 90 druhů plastů a probíhá zde provoz na tři směny. Rozvrhování výroby je potřebné realizovat na měsíc až tři měsíce dopředu. V tomto kontextu se jedná o sestavení rozvrhu pro 100 až 200 forem na jednotlivé stroje na jeden měsíc. Každý rozvrh musí splňovat nebo obsahovat následující podmínky:

Dle formy (výrobku):

- potřebná uzavírací síla stroje
- potřebná velikost stroje (gramáž)
- výrobní čas výrobku
- čas pro výměnu formy a seřízení stroje, spolu s nutností sušení materiálu
- nutnost ručních úprav a kontrola kvality výlisku a z toho plynoucí náročnost obsluhy, případná možnost paralelní obsluhy na více strojích

Dle stroje:

- možnosti gramáží a uzavírací síly
- preference podle úspornosti, modernosti, spolehlivosti
- další podmínky nasazení jednotlivých forem na stroje vycházející například ze zkušeností, apod.

Další doplňující podmínky:

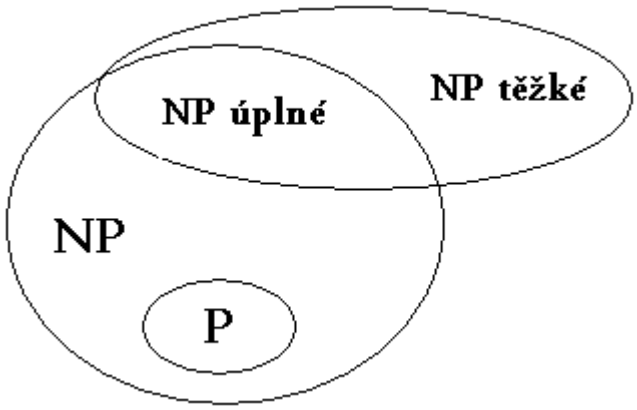
- priority po sobě jdoucích materiálů a barev
- deadline pro jednotlivé práce
- priority podle velikosti série, dobré ceny, dobře platícího zákazníka, případné sankce za nedodání v termínu, apod.
- upřednostnění konce prací na stroji před víkendem nebo svátkem

3 SLOŽITOST

Pro řešení výpočetních úloh pomocí výpočetní techniky byl navržen nástroj pro určení efektivnosti a rychlosti jednotlivých algoritmů. K tomuto účelu se používájí pojmy asymptotické složitosti a operační náročnosti algoritmu.

Asymptotická složitost algoritmu závisí na velikosti vstupních dat a podle toho ji můžeme měřit. V závislosti na změně velikosti vstupních dat se nám bude měnit chování algoritmu. Algoritmy dělíme na algoritmy s polynomiální a nepolynomiální časovou složitostí. Zapsat ji můžeme pomocí „*velké O notace*“ jako *O(f(N))* (např. *O(N)*). V tomto případě se jedná o lineární složitost. Zápis jednoduše vyjadřuje, že čas provádění algoritmu se zvýší přibližně tolikrát, kolikrát se zvětší vstupní data. U složitosti $O(N^2)$ se již bude doba zvyšovat kvadraticky.

Algoritmy je možné rozdělit do tříd složitosti. Nejzákladnější třídou složitosti je třída *P*, je obvykle považována za třídu problémů, které jsou efektivně řešitelné v polynomiálním čase na Turingově stroji, respektive na počítači. Často jsou tyto typy algoritmů nazývané efektivní nebo jednoduše dobré. Zobecněná nadmnožina je třída *NP* (zkratka nedeterministický polynomiální) *těžkých* problémů. Jedná se o problémy, které lze řešit v polynomiálně omezeném čase na nedeterministickém Turingově stroji. Nejsložitější problémy z *NP* problému se nazývají *NP úplné*. Hlavním důvodem jejich zajímavosti je právě jejich složitost. Na *Obr. 1* je možné vidět, jak spolu jednotlivé třídy souvisí. [6]



Obr. 1 Třídy složitosti [6].

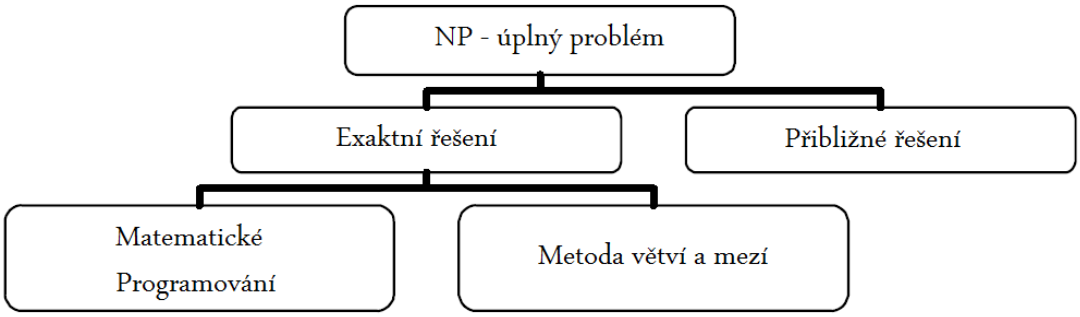
3.1 Typické příklady časové složitosti

- *O(1)* indexování prvku v poli
- $O(\log_2 N)$ vyhledání prvku v seřazeném poli metodou půleného intervalu
- *O(N)* vyhledání prvku v neseřazeném poli lineárním (sekvenčním) vyhledáváním
- *O(N log N)* seřazení pole reálných čísel podle velikosti (Algoritmem Mergesort)
- $O(N^2)$ diskrétní Fourierova transformace (DFT)
- $O(2^N)$ přesné řešení problému obchodního cestujícího (či jiného NP-úplného problému hrubou silou)

[6]

3.2 Složitost v oblasti plánování

Kapitola 4. se věnuje plánování, jehož výpočetní složitost se odvíjí od počtu strojů, prací a časů jako jsou čas začátku nebo posledního možného ukončení operace. Z pohledu časové složitosti algoritmu pro sestavení řešitelného plánu to budou hlavní ukazatelé náročnosti takového algoritmu.

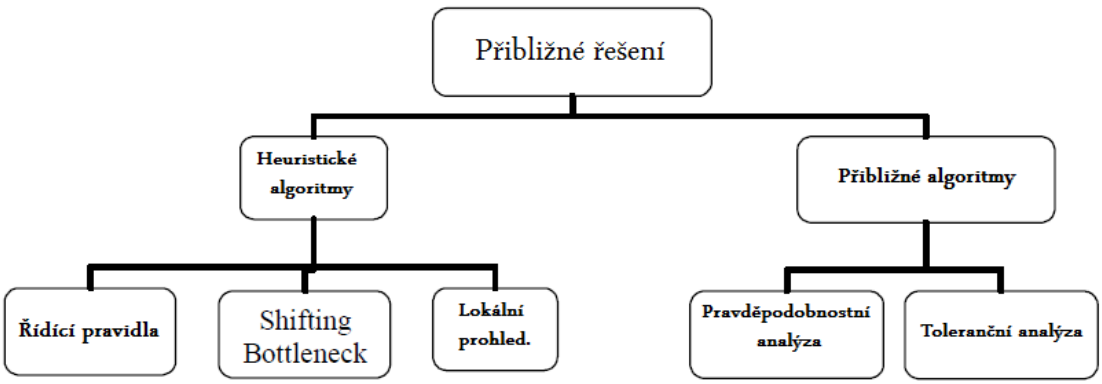


Obr. 2 Řešení NP úplného problému [2].

V zásadě tyto algoritmy navržené pro plánování obsahují základní operace jako je přičítání, odečítání, násobení a porovnávání. Z těchto důvodů je hodně rozvrhovacích algoritmů realizováno s polynomiálním časem. Typické hodnoty časové složitosti jsou $O(N^2)$ a $O(N^3m)$. Mnoho rozvrhovacích úloh v sobě zahrnuje přeskládávání n úloh, proto jsou algoritmy často se složitostí $O(N \log N)$.

Nicméně v oblasti rozvrhování pořád zůstává spoustu problémů, které sebou nesou algoritmy s *NP úplností*, pro tyto problémy se používají jiné než exaktní metody pro jejich výpočet, respektive pro sestavení plánu, rozvrhu, *Obr. 3*. Komplexní klasifikace rozvrhovacích problémů a jejich statut složitosti je možné najít ve zdroji [3].

Pokud zjistíme, že náš problém je *NP těžký, úplný*, musíme se rozhodnout, zda jsme schopni dosáhnout exaktního řešení v reálném čase nebo nám bude stačit řešení aproximační, blížící se exaktnímu. Jak je patrné z *Obr. 2,* exaktní řešení je možné také najít pomocí metod, které redukují počet výpočetních operací jako je například metoda „Větví a mezí“. Rozvrhovací problémy jsou také často dobře řešitelné pomocí celočíselného lineárního nebo nelineárního programování. Nicméně i tak je možné tyto techniky použít pro problémy s omezenou velikostí vstupních dat. Například pro *Job shop* problém bylo neznámé exaktní řešení s 10 stroji a 10 pracemi po dobu 25 let. Dnes můžeme říci, že tyto techniky jsou použitelné pro problémy do 100 prací.



Obr. 3 Dosažení přibližného řešení. [2]

Pro větší problémy je nutné použít techniky, které nezaručují nalezení exaktního řešení, proto se jim říká „aproximační algoritmy“ - jsou schopny najít jen přibližné řešení. Přesto jsou tyto techniky velmi účinné a v praxi často využívané. Na obrázku *Obr. 3* jsou patrné různé techniky, které je možné použít. Mezi tyto techniky patří řídící pravidla a různé druhy lokálního prohledávání jako jsou Horelezecký algoritmus, Tabu prohledávání, Simulované žíhání, Genetické algoritmy a další. [2]

4 PLÁNOVÁNÍ

Jako samostatné odvětví operačního výzkumu a jako věda o rozvrhování se plánování (scheduling) začalo objevovat na počátku roku 1950. Rozvrhování je používáno v plánování výroby, rozvrhování počítačových zdrojů, zemědělství, zdravotnictví, dopravě a v mnoha dalších oborech. Teorie je založena na efektivním rozvržení aktivit na jeden nebo více zdrojů v časovém sledu. Tato kapitola čerpá ze zdrojů [2].

4.1 Základní pojmy

Problém strojového plánování může být popsán následovně.

- Stroje (zdroje, prostředky) $i = 1 .. m$
- Úlohy (práce, aktivity) $j = 1 .. n$
- *(i, j)* provedení úlohy *j* na stroji *i*

- Statické parametry úlohy
  - *doba trvání* p_{ij}, p_j - doba provádění úlohy *j* na stroji *i*
  - *termín dostupnosti j* (release date) r_j - nejdřívější čas, ve kterém může být úloha *j* prováděna
  - *termín dokončení* (due date) d_j - čas, do kdy by měla být úloha *j* nejpozději dokončena (preference) vs. deadline - čas, do kdy musí být úloha *j* nejpozději dokončena (požadavek)
  - *váha* w_j : důležitost úlohy *j* relativně vzhledem k ostatním úlohám v systému

- Dynamické parametry úlohy
  - *čas startu úlohy* (start time) S_{ij}, S_j : čas zahájení provádění úlohy *j* na stroji *i*
  - *čas konce úlohy* (completion time) C_{ij}, Cj : čas, kdy je dokončeno provádění úlohy *j* na stroji *i*
  - čas zpoždění $L_j = C_j - d_j$
  - čas dřívějšího dokončení $Tj(S) = max\{C_j - d_j, 0\}$
  - pokuta za zpoždění $U_j(\sigma) = 1$ jestliže $C_j(\sigma) > d_j$, a jinak $U_j(\sigma) = 0$

- Standardní podmínky
  - jedna práce nemůže být zpracovávana dvěma nebo více stroji zároveň
  - jeden stroj nemůže zpracovávat dvě nebo více prací zároveň

- Kritéria optimality - mějme daný rozvrh *S*, pak můžeme uvažovat následující kritéria
  - maximální čas konce úlohy $C_{max} = max_j C_j$
  - maximální zpoždění $L_{max} = max_j L_j$
  - uvažujme hodnotící funkci $f_j(S) = f_j(C_j(S))$, pak maximum $f_{max} = max_j f_j$
  - celkový čas (dle váhy) procesu $\sum_j (wj)Cj$
  - celkový čas (dle váhy) dřívějšího dokončení $\sum_j (w_j)T_j$
  - celkový počet (dle váhy) zpožděných prací $\sum_j (w_j)U_j$
  - celkové náklady $\sum_j f_j$
  - další..

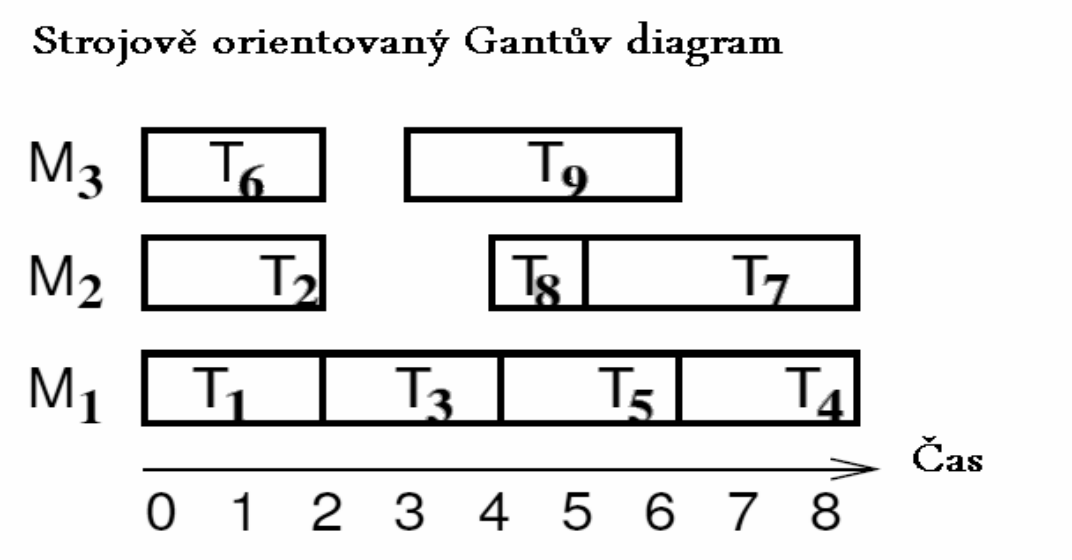
Proveditelný optimální rozvrh musí splňovat všechna zmíněná kritéria, další podmínky závislé na konkrétním případu a také kritéria optimality, která jsou uvedena níže. Na *Obr. 4* je možné vidět typický způsob reprezentace rozvrhu.

Vzhledem k tomu, že je myslitelných mnoho variant plánovacích problémů, byl přijat jednotný systém pro popis těchto problémů Graham, Lawler, Lenstra & Rinnooy Kan (1979). Jedná se o tři pole $\alpha \mid \beta \mid \gamma$, která označují typ problému. Symbol α představuje charakteristiky strojů, symbol β definuje charakteristiky úloh a symbol γ definuje kritéria optimality.

Příklady Grahamovy klasifikace:

- $P3|prec|C_{max}$ - montáž kola
- $P_m|r_j|\sum w_j C_j$ - paralelní stroje

[2], [4]



Obr. 4 Reprezentace rozvržení úloh na stroje [4].

4.2 Základní typy úloh

Stroje a jednotlivé rozvržení úloh na tyto stroje může být v několika myslitelných variantách a na tomto základě rozlišujeme několik základních typů případů. Jedna úloha je vykonána jednou na jednom stroji nebo jedna úloha je vykonávána postupně na více strojích. Dále je zde možnost kombinace jednoho stroje nebo více paralelních strojů.

- *jeden stroj* 1: 1 | .. |

- *identické paralelní stroje* P_m
  - kde čas zpracování není závislý na přiřazení úlohy na stroj
  - m identických strojů zapojených paralelně
  - úloha může být provedena na libovolném z m strojů

- *jednotlivé paralelní stroje s různou rychlostí* Q_m
  - doba trvání úlohy j na stroji i přímo závislá na jeho rychlosti v_i
  - $p_{ij} = p_j / v_i$

- *nezávislé paralelní stroje* R_m
  - stroje mají různou rychlost pro různé úlohy
  - stroj i zpracovává úlohu j rychlostí v_{ij}
  - $p_{ij} = p_j / v_{ij}$

Pro systémy, kde jedna úloha může být postupně zpracována na více strojích, máme multioperační (shop) problémy (vzhledem k neustálenosti České terminologie v této oblasti, ponechám některé pojmy nepřeloženy). Jedná se o klasické problémy, které jsou předmětem zájmu operačního výzkumu. V praxi máme často mnohem složitější problémy, které se skládají z těchto základních.

- jedna úloha je prováděna postupně na několika strojích
  - úloha *j* se skládá z několika operací *(i, j)*
  - operace *(i, j)* úlohy j je prováděna na stroji i po dobu pij
  - příklad: úloha j se 4 operacemi *(1, j); (2, j); (3, j); (4, j) Obr. 4*

Obr. 5 Výrobní postup úlohy j [4].

- *Flow shop* - všechny úlohy mají stejný pracovní postup $M_1, M_2, .. , M_m$
- *Open shop* - pracovní postup jednotlivých úloh není přesně dán a je součástí nalezeného řešení
- *Job shop* – každá práce má fixní pracovní postup, ale může být různý než u ostatních, některé stroje zde mohou chybět, některé stroje mohou být v plánu více než jednou

[2], [4]

4.3 Přiblížení některých algoritmů

a) Algoritmy pro jeden stroj

V praxi se jedná o jeden z často se vyskytujících případů, kdy máme v systému jen jednu službu, na které se musí vystřídat postupně všechny úkony. Může se jednat například o muže, který chce prostudovat celou knihovnu a složit z těchto získaných informací zkoušky v určitých časech. V jednom čase nemůže číst více jak jednu knihu, a tak mu nezbývá než postupně přečíst všechny knihy jednu po druhé. Samozřejmě může začít číst jednu, pak začít jinou a zase se k té předchozí vrátit. Rozvržení takového rozvrhu je determinováno termíny jednotlivých zkoušek. Úspěšnost na zkoušce a výsledná známka může být dána právě správným rozvržením studijního času.

Problém	Výsledek
$1 \mid\mid \sum w_j C_j$	$O(n \log n)$ WSPT
$1 \mid\mid \sum L_{\max}$	$O(n \log n)$ EDD
$1 \mid\mid \sum U_j$	$O(n \log n)$ Moore's Algorithm
$1 \mid\mid \sum w_j U_j$	NP-hard
$1 \mid\mid \sum w_j U_j$	$O(n\tau)$ DP algorithm

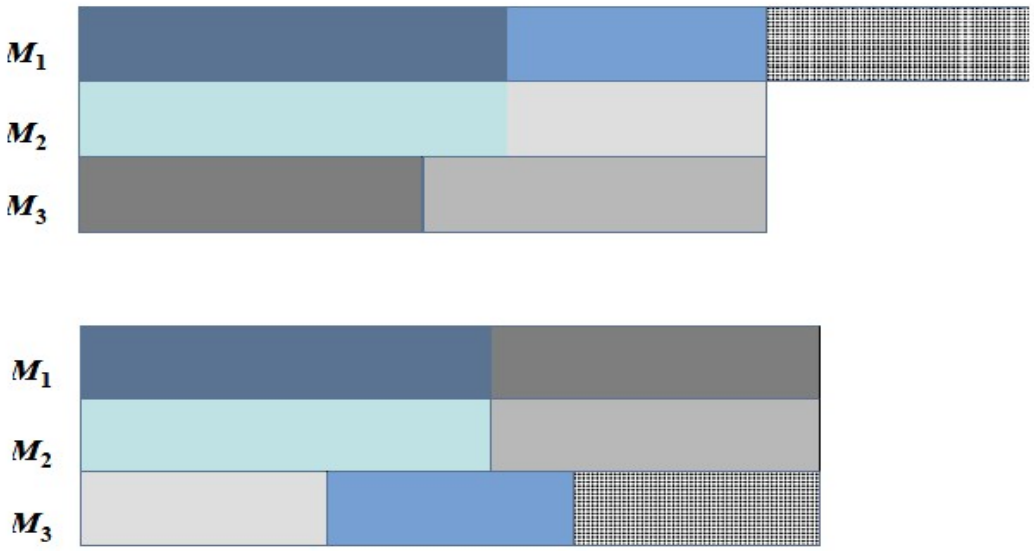
Tab. 1 Přehled algoritmů [2].

Algoritmy pro obsluhu jednoho stroje jsou základem pro algoritmy s mnoho strojovou úlohou. Jak je patrné z tabulky *Tab. 1,* je možné definovat několik základních algoritmů podle toho, které kritérium optimality zvolíme. Z tabulky také vidíme, že již u jednoho stroje najdeme *NP těžký* problém. Například *1| |ΣwjCj* , tedy minimalizace celkového času dokončení (Completion time) všech prací využívá při zpracování *WSPT* pravidlo (weighted shortest processing time), které říká, že první

bude v rozvrhu práce s nejkratším procesním časem, za ní pak s druhým nejkratším, atd. Každý algoritmus z uvedené tabulky využívá jiné pravidlo, které se k provedení dané minimalizace objektivní funkce nejvíce hodí.

b) Algoritmy pro paralelní identické stroje

Jak je vidět z *Obr. 6,* je možné seřadit úlohy na identické stroje různými způsoby a ne všechny jsou optimální.



Obr. 6 Porovnání dvou různě poskládaných rozvrhů [2].

c) Algoritmy pro jednotlivé paralelní stroje s různou rychlostí

V zásadě se jedná o podobný případ jako předchozí, jen procesní časy úloh na různých strojích jsou jiné, jelikož každý ze strojů má jinou rychlost. Například $Q_m|\ |\Sigma Cj$ nám popisuje problém, kde každá z *j* úloh musí být přiřazena jednomu stroji. Stroj M_i má rychlost s_i, takže úloha *j* na tomto stroji zabere p_j/s_i a jde nám o minimalizaci celkového času dokončení všech úloh.

Problém	Výsledek
$P2 \mid\mid C_{\max}$	NP-hard
$Pm \mid\mid C_{\max}$	Approximation ratio $2 - 1/m$ (Algorithm LS)
$Pm \mid\mid C_{\max}$	Approximation ratio $4/3 - 1/(3m)$ (Algorithm LS with LPT)
$Qm \mid\mid \sum C_j$	$O(n \log n)$
$Pm \mid\mid \sum C_j$	$O(n \log n)$ (Algorithm LS with SPT)
$P2 \mid\mid \sum w_j C_j$	NP-hard

Tab. 2 Porovnání jednotlivých základních případů [2].

d) Nezávislé paralelní stroje

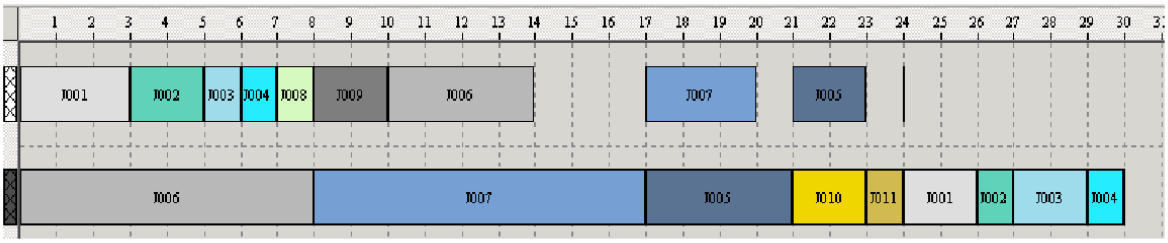
Představme si, že máme problém o *n* úlohách, které mají být rozvrženy na *m* nezávislých paralelních strojích bez priorit daných úloh. Každá úloha musí být přiřazena přesně jednomu stroji. Tímto úloha získá svůj procesní čas p_{ij}. Garey a Johnson (1979) ukázali, že problém je *NP těžký* již pro jednodušší problém s identickými stroji $R||C_{max}$, kde procesní čas úlohy nezávisí na tom, na který stroj je úloha přidělena. Při využití kombinace exaktního algoritmu a metody větví a mezí je daný problém neřešitelný již při rozvržení více jak 50 úloh na 5 strojích. [5]

e) Multioperační (shop) problémy

Zamysleme se nad tímto příkladem, který reprezentuje klasický *Job shop* problém. *J2*| $|C_{max}$, kde každá z úloh má nejvíce dvě operace. Množina všech úloh může být rozdělena na podmnožiny následujícím způsobem:

- N_A množina úloh, které budou prováděny na stroji A
- N_B množina úloh, které budou prováděny na stroji B
- N_{AB} množina úloh, které budou prováděny na stroji A a pak B
- N_{BA} množina úloh, které budou prováděny na stroji B a pak A

K řešení tohoto problému se používá tzv. Jackson algorithmus, implementační detaily je možné najít v použité literatuře [2]. Na *Obr. 7* se můžeme podívat na rozvrh sestavený tímto algoritmem.



Obr. 7 reprezentace rozvrhu vytvořená Jackson algorithmem [2].

K řešení dalších typů úloh se používá celá řada algoritmů. V tabulce *Tab. 3* je možné vidět přehled základních problémů a jejich časové složitosti. [2], [4]

Problém	Výsledky
$F2 \mid\mid C_{\max}$	$O(n \log n)$ (Johnson's algorithm)
$F3 \mid\mid C_{\max}$	NP-hard in the strong sense
$J2 \mid\mid C_{\max}$	$O(n \log n)$ (Jackson's algorithm)
$J \mid\mid C_{\max}$	NP-hard
$O2 \mid\mid C_{\max}$	$O(n)$ (Three algorithms)
$O3 \mid\mid C_{\max}$	NP-hard
$Fm \mid\mid C_{\max}$	Approximation Algorithms
$Fm \mid\mid \sum C_j$	Approximation Algorithms
$Om \mid\mid C_{\max}$	Greedy Approximation Algorithms

Tab. 3 Přehled základních případů multioperačních problémů [2].

5 HEURISTICKÉ METODY

Etymologie slova heuristika spadá do Starého Řecka (z řečtiny *heuriskó*, *εύρίσκω* – nalézt, objevit). V novodobém smyslu znamená heuristika zkusmé řešení problémů, bez znalosti přesnějšího algoritmu nebo metody. Takové řešení je často jen přibližné a založené na odhadu, intuici, zkušenosti nebo zdravém rozumu (common sense). V kontextu informatiky mluvíme o tzv. heuristickém postupu, který sice nedává přesné řešení daného problému, ani nám nemůže zaručit nalezení řešení v krátkém čase, ale v mnoha případech je schopen najít řešení dostatečně přesné a i v relativně krátkém čase. Toto tvrzení však nelze dokázat, obecně se heuristické postupy používají tam, kde neznáme lepší exaktní algoritmus pro nalezení přesného řešení. Tyto metody tedy nezaručují nalezení globálně optimálního řešení. Typickým příkladem aplikace heuristických postupů jsou *NP úplné* úlohy jako například Problém obchodního cestujícího (Travelling salesman problem), ve kterém jde o nalezení nejkratší cesty procházející všemi body na mapě. [7], [8], [9]

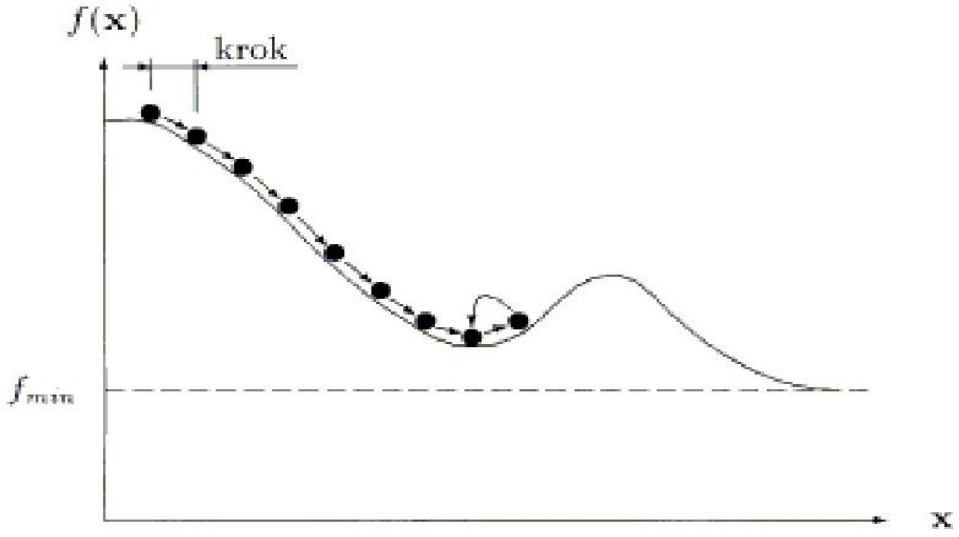
Heuristických metod je celá řada, tato kapitola se bude zaměřovat pouze na tzv. stochastické heuristické optimalizační metody, které si svou stochastičnost zachovávají v celém průběhu optimalizačního procesu.

Abychom mohli posoudit, jak je které řešení kvalitní, musíme zavést tzv. hodnotící funkci, která je schopna ohodnotit jakékoliv řešení z prostoru všech možných řešení daného problému. Tato funkce se nazývá účelová nebo taktéž cenová funkce. Z toho tedy plyne, že heuristické metody mohou být nasazeny jen tam, kde je možné takovou funkci sestrojit. Dle vztahu (1) vidíme, že funkce je zobrazením z množiny všech řešení $\boldsymbol{X}$ do množiny reálných čísel. [10].

$$f(\boldsymbol{x}): \mathbf{X} \rightarrow \mathbf{R} \qquad \textbf{(1)}$$

5.1 Horolezecký algoritmus

Jedná se o gradientní algoritmus bez nutnosti vyjádřit gradient exaktně (Hill climbing). Směr nejprudšího spádu a tedy i pokračování prohledávání se určí na základě generování všech sousedů v okolí aktuálního řešení. Algoritmus je ukončen, pokud žádný ze sousedů nedosahuje lepšího ohodnocení *Obr. 9*. Takto se často stane, že algoritmus uvázne v nejbližším lokálním optimu, které je prohlášeno za výsledné optimum. Tato nevýhoda se může kompenzovat náhodným spuštěním algoritmu z různých míst stavového prostoru. V *Tab. 3* je uveden pseudokód horolezeckého algoritmu.



Obr. 8 Ukázka zacyklení algoritmu [9].

Další možnou modifikací algoritmu je náhodné generování jen části všech sousedů aktuálního řešení nebo zastavení výpočtu po určitém počtu iterací. [6], [10]

```
Horolezecký algoritmus
  současnýUzel = startovacíUzel;
  opakuj
    Seznam = Sousede(současnýUzel);
    dalsiOhodnoceni = -INF;
    dalsiUzel = NULL;
    foreach x in Seznam
      if (EVAL(x) > dalsiOhodnoceni)
          dalsiUzel = x;
          dalsiOhodnoceni = EVAL(x);
      if dalsiOhodnoceni <= EVAL(současnýUzel)
         //Vrať současný uzel, pokud neexistuje lépe ohodnocený soused
          return současnýUzel;
      současnýUzel = dalsiUzel;
```

Tab. 4 Pseudokód Horolezeckého algoritmu

5.2 Zakázané prohledávání

Je obdobou předešlého algoritmu, vylepšený o seznam s historií předešlých řešení, tzv. zakázaných řešení (Tabu search). Do seznamu s historií je možné umístit určitý počet řešení. Při inicializaci algoritmu je seznam zakázaných řešení prázdný. Myslíme tím tedy, že seznam stanovený velikostí k je po k iteracích zaplněn a při další iteraci je poslední prvek - v našem případě zakázané řešení - ze seznamu vymazán, a je do něho přidán aktuální prvek - v našem případě zakázané řešení pro následující iteraci.

Na základě tohoto seznamu je možné vyřešit problém Horolezeckého algoritmu a jeho uvíznutí v blízkém lokálním optimu počátečnímu řešení. Parametr k, tedy velikost seznamu zakázaných řešení, hraje významnou roli při úspěšnosti algoritmu. Při volbě malého seznamu se algoritmus redukuje na Horolezecký algoritmus, naopak při volbě velkého seznamu mohou být přeskočena jinak nadějná řešení v závislosti na konkrétní podobě algoritmu.

Další možnou modifikací algoritmu je zaznamenávání určitých atributů řešení a jejich zákaz u následujících řešení. Tímto se může výrazně zvýšit diverzita (rozmanitost) hledaných řešení směrem ke globálnímu optimu. [6], [10]

5.3 Simulované žíhání

Metoda Simulovaného žíhání má v sobě ze zatím uvedených metod nejvíce stochastičnosti. Metoda přijímá na základě určité pravděpodobnosti (dané Metropolisovým kritériem) i horší řešení. Na rozdíl od Horolezeckého algoritmu vzorkuje celý prohledávaný prostor tím, že nevybírá nejlepší řešení z aktuálního okolí, ale určitým operátorem se stochasticky transformuje výchozí řešení x na nové x'.

Inspirací metody je fyzikální děj při žíhání tuhého tělesa, u kterého dojde k odstranění defektů krystalické mřížky. Při vysoké teplotě dochází k zániku defektů krystalické mřížky a následným pomalým ochlazením se vyloučí možnost vzniku nových defektů. Při žíhání se soustava snaží dostat do stavu s minimální energií, tedy do stavu bez defektů. Právě k tomuto procesu vede jistá analogie k řešení optimalizačních problémů. Na tuto myšlenku přišli na počátku 80. let minulého století výzkumní pracovníci z výzkumného centra IBM (Watson Research Center of the IBM, USA). Proces žíhání bylo nutné nahradit numerickou reprezentací. K tomuto účelu bylo použito Metropolisovo kritérium (2).

$$P(x \to x') = \begin{cases} 1 & \text{pro } f(x') \le f(x) \\ e^{\frac{f(x')-f(x)}{T}} & \text{pro } f(x') > f(x) \end{cases} \quad (2) \qquad [13]$$

Jak je vidět, nový stav je přijmut, pokud je hodnotící funkce nového stavu stejná nebo větší než je hodnotící funkce předešlého stavu. Nový stav s větší hodnotící funkcí je přijat s pravděpodobností v rozmezí $0 < P(x \to x') =< 1$. Podle velikosti T se nám mění pravděpodobnost přijetí nového stavu. Pro velké hodnoty T se pravděpodobnost blíží 1, akceptují se všechny nové stavy. S klesající T se pravděpodobnost blíží 0, tímto se jen výjimečně akceptuje stav s vyšší hodnotou hodnotící funkce. Tento postup lze interpretovat tak, že na počátku procesu dochází k velkým skokům po stavovém prostoru, při snižování teploty a parametru T dochází k dolaďování nalezeného řešení. *Tab 5.* nám reprezentuje pseudokód Metropolisova kritéria, kde *Opert* je stochastický operátor poruchy aktuálního stavu na následující.

```
vector Metropolis(vector x0, unsigned int kmax, double T)
{
        vector x, x';
        double p;
        x = x0;
        for (unsigned int k = 0 ; k < kmax ; k++) {
                x' = Opert(x);
                p = min(1, exp(-( f(x') - f(x) ) / T));
                if (uniform(0,1) < p)
                x = x';
        }
        return x;
}
```

Tab. 5 pseudokód Metropolisova kritéria [13].

Metropolisův algoritmus se opakuje k_{max} krát, vektor x_0 je počáteční stav, pokud se k_{max} blíží nekonečnu, generuje tento algoritmus rozložení blízké Boltzamnovu rozložení. Tato důležitá vlastnost je právě využita k numerické simulaci procesu žíhání. Simulované žíhání je pak proces, při kterém dochází ke snižování teploty T a opakovaného využití Metropolisova algoritmu.

Výsledný stav z každé iterace je použit jako výchozí stav pro další snížení teploty. To, jakým způsobem dochází ke snižování teploty, záleží na řešeném problému a mluvíme zde o tzv. plánu ochlazování. Snižování teploty je možné realizovat skokově po diskrétních skocích nebo pomocí koeficientu $0 << a < 1$, tzn. $T = aT$, v praxi se často nastavuje hodnota parametru alfa mezi 0.8 a 0.99. Další možnou metodou je využití genetického algoritmu při určování nové teploty, který dokáže pružněji reagovat na konkrétní situaci.

Parametry k_{max}, T_{max}, T_{min} a jsou základní konstanty algoritmu, teplota T_{max} se postupně snižuje až na teplotu T_{min}. *Tab. 6* nám reprezentuje pseudokód algoritmu, můžeme si všimnout, že Metroplisův algoritmus je volený při každém snížení teploty.

Realizace Simulovaného žíhání sebou přináší některé praktické problémy, a to je způsob, jakým se nastaví jednotlivé parametry. Ty totiž přímo ovlivňují průběh a úspěšnost samotné metody. Správné nastavení těchto parametrů nelze odpovědět obecně, protože se liší tím, na jaký problém je metoda aplikována. Pro počet iterací k_{max} se doporučuje hodnota někde mezi 10^3 - 10^5. Počáteční a koncová teplota se musí zvolit tak, aby nebyly akceptovány všechny nově generované stavy, ale zase aby jich nebylo akceptováno příliš málo. Při akceptování všech porušených (nových) stavů se metoda redukuje na slepé prohledávání stavového prostoru. [10].

```
vector Simulated_annealing(double Tmin, double Tmax, unsigned int kmax, double α)
{
      vector x0, xmin;
      double T = Tmax;
      x0 = náhodne vygenerovaný stav;
      while (T > Tmin) {
            x0 = Metropolis(x0, kmax, T);
            T = α*T;
      }
      return x0;
}
```

Tab. 6 Pseudokód simulovaného žíhání [13]

6 GENETICKÉ ALGORITMY

Tak jako metody z 5. kapitoly je Genetický algoritmus heuristický postup, který je dobře využitelný tam, kde neexistuje přesný algoritmus pro nalezení exaktního řešení. Myšlenka, na které jsou tyto přístupy založeny, pochází z evoluční biologie, kdy se snažíme aplikací technik napodobující evoluční procesy jako je dědičnost, přirozený výběr, mutace, křížení a jiné dosáhnout řešení složitých problémů. Tyto přístupy se začaly objevovat na počátku 60. let minulého století a pocházejí z Darwinovy teorie o vývoji druhu. Teoretické aspekty vhodné pro aplikaci do oblasti informatiky byly rozvedeny v knize *Adaptation in Natural and Artificial Systems,* kterou napsal v roce 1975 John Henry Holland a obecně se považuje za zakladatele v této oblasti. Tyto techniky můžeme dělit následovně:

- Genetické algoritmy
- Genetické programování
- Evoluční strategie
- Evoluční programování
- Gramatická evoluce

[11]

6.1 Postup Genetického algoritmu

1. Inicializace populace – nová populace je obvykle složena z náhodně vygenerovaných jedinců
2. Pomocí určité výběrové metody jsou z populace vybrání jedinci podle určitého kritéria, většinou na základě jejich zdatnosti
3. Z vybraných jedinců jsou generováni jedinci noví pomocí následujících metod:
   - křížení – prohození částí jedinců mezi sebou
   - mutace – náhodná změna části jedince
   - reprodukce, elitismus – kopíruj jedince do další populace beze změny
4. Výpočet zdatnosti jednotlivých nově vytvořených jedinců
5. Opakuj od bodu dva, pokud není splněna některá ukončovací podmínka
6. Pokud je splněna ukončovací podmínka, vezmi nejzdatnějšího jedince a pošli ho jako výstup programu

[11]

6.2 Základní pojmy

Genotyp (Genotype) - obecná genetická informace
Fenotyp (Phenotype) - konkrétní případ reprezentace genetické informace
Chromozóm (Genome) - obecná genetická informace reprezentována sekvencí symbolů z nějaké abecedy (reálná čísla, znaky nebo jejich kombinace, apod.)
Jedinec (Individual) - nositel genetické informace
Gen (Gene) - místo (pozice) v chromozómu
Alela (Allele) - konkrétní symbo, kterého nabývá gen v chromozómu
Rodič (Parent) - jedinec vybraný metodou selekce, následně vstupující do rekombinace
Potomek (Offspring) - jedinec, který vznikne na základě rekombinace dvou nebo více potomků
Ohodnocení (Fitness) - ohodnocení jedince, které rozhoduje o jeho přežití

Křížení (Crossover) - ze dvou nebo více jedinců, vznikne nový jedinec nebo jedinci, jejich kombinací
Mutace (Mutation) - náhodná změna alel nebo alely genů v chromozómu
Rekombinace (Recombination) - označení pro křížení a mutaci
Selekce (Selection) - výběr jedinců pro další generaci nebo rekombinaci
Generace (Generation) - skupina jedinců, která prochází vývojem
Populace (Population) - obecnější vyjádření pro skupinu jedinců
Migrace (Migration) - přesun jedinců mezi populacemi
Doplnění, znovuvložení (Reinsertion) - přechod jedinců do další generace bez nutnosti křížení nebo mutace
Schéma (Scheme) - předpis pro chromozóm, má určené některé geny

[11]

6.3 Konstrukce jedince

a) Kódování

Způsob kódování jedince hraje významnou roli při úspěchu nebo neúspěchu Genetického algoritmu v aplikaci na konkrétní úlohu. Nejpoužívanější způsob kódování je z mnoha důvodů binární kódování. Takto vyjádřený chromozóm obsahuje geny, které nabývají hodnot 0 nebo 1. Tyto hodnoty jsou jednotlivé alely genu, tedy hodnoty, ve kterých se gen může nacházet. Například *Tab. 8* nám reprezentuje zakódování populace o dvou jedincích.

Jedinec číslo	Chromozóm jedince
1	(0, 1, 0, 0, 0, 1, 0, 1, 1, 1)
2	(1, 0, 1, 1, 1, 0, 0, 1, 0, 0)

Tab. 8 Způsob zakódování jedinců.

číslo	Binární kód	Grayův kód		číslo	Binární kód	Grayův kód
0	0000	0000		8	1000	1100
1	0001	0001		9	1001	1101
2	0010	0011		10	1010	1111
3	0011	0010		11	1011	1110
4	0100	0110		12	1100	1010
5	0101	0111		13	1101	1011
6	0110	0101		14	1110	1001
7	0111	0100		15	1111	1000

Tab. 9 Porovnání binárního a Grayova kódu.

Tento způsob kódování má jednu nevýhodu, a to je často velká změna jedince při nepatrné změně v jeho chromozómu. Pokud například jedinec představuje vzdálenost od nějakého počátku soustavy souřadnic, znamená změna bitu na vysoké pozici chromozómu sice nepatrnou změnu jedince, ale v kontextu soustavy souřadnic jde o velkou změnu ve vzdálenosti. Nevýhodu tohoto typu lze řešit na základě Grayova kódu, kde se dvě sousední hodnoty zakódované v tomto typu kódu liší jen o jeden bit. Tuto skutečnost je možné vidět v *Tab. 9* .

Obecně však lze jedince zakódovat pomocí znaků nebo reálnými čísly. Tento způsob se často používá tam, kde se pracuje s velkými čísly a binární reprezentace by tvořila příliš dlouhé řetězce. V oblasti plánování a u různých kombinatorických úloh se používá permutační kódování, kde často záleží na pořadí a jednotlivá čísla se nesmí v chromozómu opakovat. [11]

b) Hodnotící fitness funkce

Hodnocení jedinců je možné udělat na základě účelové funkce, tak jak bylo zmíněno v kapitole 5., ale i mnoha jinými způsoby. Pokud máme například dostatečně velkou populaci, je možné ji rozdělit na dvě a jedince vzájemně porovnat. [11]

c) Schéma chromozómu

Schéma chromozómu je řetězec délky *l* nad množinou symbolů pro daný chromozóm, rozšířený o zástupný symbol, který se nejčastěji označuje *. Schéma si tedy můžeme představit jako šablonu, která popisuje podmnožinu symbolů dané délky *l*. Schéma tedy obsahuje určené a dosud neurčené pozice v chromozómu. Říkáme, že řetězec je instancí schématu, pokud se shodují všechny symboly řetězce a schématu, které jsou ve schématu určené. Počet určených pozic je definován jako řád schématu. Schémata jsou důležitým nástrojem při analýze a chování Genetického algoritmu. [11]

Schéma č. 1	Schéma č. 2
(1*0*0*1*0)	(*01**1**)
Instance	Instance
(100100110) (100101110) (110001110)	(00101110) (00110111) (00101110)

Tab. 10 Schémata a jejich možné instance.

6.4 Výběrové metody (selekce)

Tyto metody by měly imitovat proces přirozeného výběru. Z tohoto důvodu jsou do procesu reprodukce vybíráni silnější jedinci a ti se reprodukují na úkor těch slabších. Stejně jako v přírodě dochází k tomu, že potomstvo získá většinou kvalitní vlastnosti od svých rodičů. Základní principy však respektují rozmanitost, která zde musí být také zachována. Tak jako v přírodě má každý samec šanci u každé samičky a nezáleží na tom, jak kvalitní jedinec to je, tak i zde musí být tento základní princip zachován. Přílišné upřednostňování určitých jedinců - těch silnějších, by mohlo znamenat uváznutí v lokálním minimu, naopak přílišná rozmanitost (diversivita) by nezaručovala rychlou konvergenci k optimu. [11]

a) Ruletový výběr

Na počátku jsou sečteny fitness funkce jednotlivých jedinců a velikost fitness funkce každého jedince udává pomyslnou velikost výseče na ruletovém kole *Obr. 9*. Při výběru jedinců je aplikovaný náhodný výběr jedince. Pokud je jedinec vybrán, padne na něj kulička ruletového kola a postupuje do další generace. Pravděpodobnost se kterou bude jedinec vybrán, je možné popsat následovně:

$$p(i) = \frac{f(i)}{\sum_{j=1}^{N} f(j)} \qquad \textbf{(2)}$$

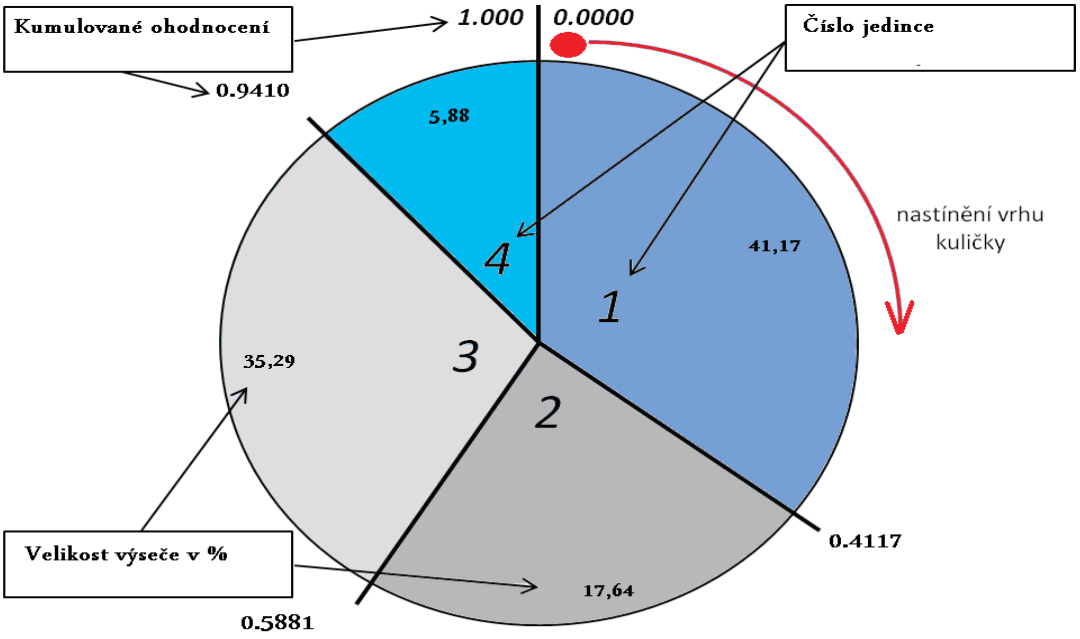
Kde $i \in \{1 \ldots N\}$, N je počet jedinců populace, $f(i)$ je ohodnocení i-tého jedince a suma $f(j)$ značí součet všech ohodnocení populace.

Pro zjednodušení umístění pomyslné kuličky na ruletovém kole zavádíme tzv. kumulativní ohodnocení *Tab. 11*, pak je možné generovat náhodné umístění kuličky v intervalu <0, 1>.

Jedinec číslo	Chromozóm jedince	Ohodnocení *f(i)*	% z celkového ohodnocení *p(i)*	Komulované Ohodnocení
1	(1, 1, 0, 0, 1, 1, 0, 1, 1, 1)	7	41.17 %	0.4117
2	(1, 1, 1, 1, 0, 1, 0, 0, 0, 0)	3	17.64 %	0.5881
3	(1, 1, 1, 0, 0, 0, 0, 1, 0, 1)	6	35.29 %	0.9410
4	(0, 0, 1, 1, 0, 0, 1, 1, 0, 0)	1	5.88 %	1.0000

Tab. 11 Kumulované ohodnocení

Takto jsou postupně vybráni noví jedinci v závislosti na tom, do kterého intervalu nám kulička padne. Představme si, že budeme generovat čísla 0.1213, 0.5522, 0.2876 a 0.9622, pak získáme dvakrát jedince číslo 1, jednou jedince číslo 2 a 4. Tito pak budou pokračovat v reprodukci.



Obr. 9 Reprezentace výsečí pomocí ruletového kola

Tento typ selekčního mechanismu favorizuje nejsilnější jedince a u některých typů konkrétních úloh může uváznout rychle v lokálním optimu. [11]

b) Stochastický univerzální výběr

Tento typ výběru je podobný předchozímu s malým rozdílem. Při výběru není generován stejný počet náhodných čísel, jako je jedinců, ale je vygenerováno jen jedno náhodné číslo z intervalu *<0, 1>*, které je následně dělené počtem jedinců. Toto číslo nám určuje prvního vybraného jedince. Ostatní jsou vybráni v po sobě jdoucích intervalech, které mají konstantní délku *1/N*, kde *N* udává počet jedinců, které chceme vybrat. Tímto je částečně vyřešena rozmanitost, metoda tolik nefavorizuje jedince s nejlepším ohodnocením. [11]

c) Turnajový výběr

Jedná se o hodně používanou metodu, hlavní výhodou je jednoduchost implemetace při zachování podmínek selekčního mechanismu. Z populace jsou vybráni jednotlivci, nemusí jít obecně jen o dva a ti jsou podrobeni souboji, jedinec s vyšším ohodnocením přežívá a postupuje do dalšího kola. Na základě této skutečnosti je splněn předpoklad, aby přežívali lépe ohodnocení jedinci, ale zároveň je splněna diverzivita, protože jedinci jsou k souboji vybíráni náhodně. Do výběru je možné zakomponovat taktéž náhodu, tedy vítězný jedinec postopí nebo ne, jak je vidět na *Tab. 12.* [11]

Jedinec číslo	Ohodnocení jedince	Jedinci vybraní do souboje	Výherce souboje	Poznámky
1	5	(1,8)	8	
2	11	(3,4)	**4**	**jedinec č. 3 měl smůlu**
3	14	(3,9)	3	
4	4	(4,9)	4	
5	2	(1,2)	2	
6	16	(2,3)	3	
7	8	(9,8)	8	7 již jednou postoupil
8	8	(5,7)	7	náhodný výběr
9	1	(2,5)	**5**	**jedinec č. 2 měl smůlu**

Tab. 12 Výběr jedinců dle soubojů

d) Podle pořadí

Tato metoda byla vytvořena za účelem potlačení vlivu nadprůměrných jedinců v populaci. Jedinci jsou seřazeni vzestupně dle jejich ohodnocení. Selekce pak probíhá na základě pozice jedince v tomto seznamu. Rozlišují se dva základní typy, lineární a nelineární selekce. Více o této metodě je možné najít zde [23].

e) Místní výběr

Každý jedinec má určeno své vlastní okolí, které nazýváme místním okolím, a může vstupovat do rekombinace pouze s jedinci z tohoto okolí. První krok metody je výběr poloviny populace. Tento výběr probíhá pomocí některé jiné selekční metody. Následně takto vybraným jedincům je definováno jejich okolí. Toto okolí je definováno s ohledem na strukturu, ve které je populace rozmístěna. Následně jsou původně vybraní jedinci rekombinováni s nějakým jedincem ze svého okolí, opět je možné použít některý ze selekčních mechanismů. Struktura okolí a její grafické znázornění a srovnání je možné najít ve zdroji [23], zde je stručný výpis:

- Lineární
  - plný kruh, poloviční kruh
- dvojdimenzionální
  - plný kříž, poloviční kříž
  - plná hvězda, poloviční hvězda
- trojdimenzionální a více komplexní s jakoukoli kombinací předchozích struktur

f) Seříznutý výběr

Tato metoda má jednoduchý charakter, vybírá ze seřazeného seznamu jedinců dle jejich ohodnocení. Vybere pouze určitý počet těch nejlepších. Metoda je vhodná pro hodně velké populace. Práh pro seříznutí trunction treshold (řezací práh) se obvykle nastavuje mezi 10 až 50 %. [23]

g) Elitismus

Zejména v malých populacích může při výběru jedinců dojít k tomu, že ti nejlepší nejsou vybráni pro rekombinaci. Z tohoto důvodu je praktické použít elitismus, který vyjme jednoho nebo více nejzdatnějších jedinců, kteří následně postupují do nové generace přímo.

6.5 Operátory křížení

Na základě křížení jsou vytvářeni noví jedinci, kteří nebyli součástí předchozí populace. Tvorba jedince probíhá tak, že z částí jedinců, kteří vstupují do rekombinace, jsou vytvářeni noví. Křížení jedinců mezi sebou by mělo respektovat zdatnost jedince dle selekčních metod popsaných v předchozí podkapitole. Takto mohou vznikat dosud nevytvořené kombinace řešení a Genetický algoritmus se takto často vymaní z partikulárního lokálního minima. Úspěšnost jednotlivých operátorů záleží na typu konkrétní úlohy. Mezi nejzákladnější operátory křížení patří binární operátory, máme je v několika variantách. [11]

a) Jednobodové křížení

Na základě selekčního mechanismu mějme vybrány dva chromozomy, pak je možné vygenerovat náhodně číslo mezi 0 a *K-1*, kde *K* reprezentuje délku chromozómu. Negenerujeme v rozmezí až *K,* protože bychom měli hranici na poslední pozici a oba chromozómy by zůstaly nezměněny. *Tab. 13* nám ukazuje proces křížení pro K = 3. [11]

Původní chromozómy (rodiče)		Nové chromozómy (potomci)
(**0**, **1**, **0**, **1**, **1**, **1**, **1**, **1**, **1**, **0**, **1**)	→	(**0**, **1**, **0**, 0, 1, 0, 1, 1, 0, 1, 1, 1)
(0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1)	→	(0, 1, 0, **1**, **1**, **1**, **1**, **1**, **1**, **0**, **1**)

Tab. 13 Jednobodové křížení pro K = 3.

b) Dvou a vícebodové křížení

Dvoubodové křížení je podobné jednobodovému, jen od druhého bodu nedochází k záměně genů. To je možné vidět v *Tab. 14.*

Původní chromozómy (rodiče)		Nové chromozómy (potomci)
(**0**, **1**, **0**, **0**, **1**, **1**, **1**, **1**, **1**, **0**, **1**)	→	(**0**, **1**, **0**, **0**, 1, 0, 1, **1**, **1**, **0**, **1**)
(0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0)	→	(0, 1, 1, 0, **1**, **1**, **1**, 1, 0, 1, 0, 0)

Tab. 14 Dvoubodové křížení pro body 4 a 7.

Podobným způsobem dochází k vícebodovému křížení. Každý lichý bod určuje pozici, od které jsou geny chromozómů prohazovány a každý sudý bod křížení pak bod, kde dojde k přerušení záměny genů, jak ukazuje *Tab. 15.* [11]

Původní chromozómy (rodiče)		Nové chromozómy (potomci)
(**1**, **0**, **0**, **0**, **1**, **1**, **1**, **1**, **1**, **1**, **0**, **0**)	→	(**1**, **0**, **0**, **0**, 1, **1**, 1, 1, 0, 1, **0**, **0**)
(0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0)	→	(0, 1, 1, 0, **1**, 0, **1**, **1**, **1**, **1**, 0, 0)

Tab. 15 Vícebodové křížení v bodech 4, 5, 6 a 10.

c) Uniformní křížení

Tento typ křížení zobecňuje dvě předchozí a používá tzv. křížící masku, která má stejnou délku jako vstupující chromozómy. Na základě této je každé místo potenciálním bodem křížení. Na všech pozicích této masky jsou vygenerovány alely s hodnotami 0 nebo 1. Tato čísla udávají, který rodičovský chromozóm předá konkrétní geny chromozómu potomka. Křížící maska druhého rodiče je inverzní ke křížící masce prvního rodiče. Podívejme se na způsob konstrukce potomka *Tab. 16.* Každý gen, který potomek získá, je náhodně vybrán s určitou pravděpodobností. První gen z prvního potomka má hodnotu genu z prvního předka, pokud je v jeho křížící masce alela prvního genu 1, pokud je v křížící masce alela 0, tak gen potomka má první gen podle genu druhého předka. [11]

Původní chromozómy (rodičovské)	Křížící maska	Nové chromozómy (potomci)
(**1**, **1**, **0**, **0**, **1**, **1**, **1**, **1**, **1**, **1**)	(1, 0, 1, 0, 1, 1, 1, 1, 1, 0)	(**1**, 0, **0**, 0, **1**, **1**, **1**, **1**, **1**, 1)
(0, 1, 0, 0, 1, 0, 1, 1, 0, 1)	(0, 1, 0, 1, 0, 0, 0, 0, 0, 1)	(0, **1**, 0, **0**, 1, 0, 1, 1, 0, **1**)

Tab. 16 Uniformní křížení.

Některé konkrétní úlohy používají permutační kódování. To znamená, že žádný z genů chromozómů nesmí mít stejnou hodnotu, nejednalo by se pak o permutaci. U předešlých technik nelze obecně zabránit tomu, aby některé hodnoty přebývaly nebo naopak chyběly. Musí se použít nějaká opravná procedura, která vytvoří správně utvořené permutace. Z tohoto důvodu se používají následující operátory.

a) Křížení s částečným přiřazením

Nejdříve se vymění podřetězce definované pomocí bodů křížení, ostatní geny zůstávají zatím neobsazené, jak je vidět v *Tab. 17,* jsou obsazeny znakem *?*. Současně s touto výměnou je potřeba uložit geny, které se výměny zúčastnily ve formě přepisovacích pravidel, pro následnou opravu nově vzniklého chromozomu. V dalším kroku se z rodičovských chromozómů přepíšou geny, které nepůsobí v nově vzniklých chromozómech konflikt a na neobsazená místa se aplikují přepisovací pravidla. [11, 24]

Rodičovské chromozómy		Výměna genů mezi body křížení		Bezkonfliktní doplnění chromozómů
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)	→	(?, ?, ?, **4**, **0**, **9**, ?, ?, ?, ?)	→	(?, **1**, **2**, 4, 0, 9, **6**, **7**, **8**, ?)
(3, 8, 7, 4, 0, 9, 5, 2, 1, 6)	→	(?, ?, ?, **3**, **4**, **5**, ?, ?, ?, ?)	→	(?, **8**, **7**, 3, 4, 5, ?, **2**, **1**, **6**)
		↑↑		
		Přepisovací pravidla:		
		4 ↔ 3, **0 ↔ 4**, **9 ↔ 5**		
Bezkonfliktní prototypy		Aplikace přepisovacích pravidel		
(?, **1**, **2**, 4, 0, 9, **6**, **7**, **8**, ?)	→	(**3**, 1, 2, 4, 0, 9, 6, 7, 8, **5**)		
(?, **8**, **7**, 3, 4, 5, ?, **2**, **1**, **6**)	→	(**0**, 8, 7, 3, 4, 5, **9**, 2, 1, 6)		
		↑↑		
		Použitá pravidla:		
		4 ↔ 3, **0 ↔ 4**, **9 ↔ 5**		

Tab. 17 Postup při vytváření nových chromozómů.

b) Křížení s rekombinací hran

Tento operátor je dobře aplikovatelný na problém obchodního cestujícího. Vychází ze dvou rodičovských chromozómů, k jednotlivým genům těchto chromozómů se vypíšou jejich sousedé. Každý gen tak získá seznam 4 genů. Následně konstrukce nového jedince probíhá tak, že pro každý gen jsou náhodně vybráni jeho sousedi z jeho seznamu sousedů. Přednost má ten soused, který má svůj seznam sousedů nejkratší. První gen nového chromozomu je náhodně vybrán. Po umístění genu do tabulky nového chromozomu dojde k odstranění tohoto genu z tabulky genů a také ze všech seznamů sousedů u jednotlivých genů. Ze dvou rodičovských genů tak vznikne jen jeden potomek. [24]

Poslední doménou operátorů křížení jsou jedinci obsahující reálná čísla. Tuto problematiku je možné najít v použitém zdroji [23]. Mezi tyto operátory patří Střední křížení, Líniové křížení a Rozšířené líniové křížení.

6.6 Operátory mutace

Mutace přináší do nově vzniklých chromozómů po křížení prvek náhodné změny. Tak jako v přírodě nejsou kopie potomků po svých rodičích přesnými obrazy, ale některé geny při rekombinaci změní náhodně svou informaci. Na základě tohoto operátoru se můžeme dostat v prohledávaném prostru k řešením, která nejsou na základě křížení dosažitelná. Mutace může také pomoci vylepšit tu populaci, kde je například jen jeden silný jedinec a zbytek jedinců je slabších.

a) Binární mutace

Při tomto druhu mutace dochází u binárního kódování ke změně genů z 1 na 0 nebo opačně. *Tab. 18* reprezentuje 5 % možnost změny u každého genu.

Chromozóm před mutací:	(1, 1, 0, 0, 1, 0, 1, **1**, 1, 1)
Chromozóm po mutaci:	(1, 1, 0, 0, 1, 0, 1, **0**, 1, 1)

Tab. 18 Binární mutace.

Četnost mutace je možné zvyšovat při stagnující populaci a při následném zlepšení opět vrátit na původní hodnotu. Je také možné mutaci při inicializaci nastavit na vyšší hodnotu a s počtem iterací ji snižovat. To lze chápat tak, že nejdříve prohledáváme stavový prostor do šířky a později se spíše snažíme vylepšit nadějná řešení. [11]

b) Reálné a celočíselné operátory mutace

Existuje celá řada operátorů mutace s reálnými nebo celočíselnými hodnotami. Patří mezi ně:

- Náhodná výměna – nahrazení původní alely náhodnou hodnotou
- Aditivní změna – přičtení nebo odečtení náhodně vygenerované hodnoty k původní alele.
- Multiplikativní změna – vynásobení nebo vydělení hodnoty náhodně vygenerovaným číslem.

c) Výměnná mutace

Tento typ mutace je dobře použitelný u permutačního kódování, kde by náhodná změna alely mohla znamenat opakování hodnot v chromozómu a navíc by původní číslo vypadlo. Proto je možné použít následující techniku, kdy se na náhodně zvolených pozicích dva geny vymění *Tab. 19.* [11]

Původní chromozóm		Zmutovaný chromozóm
(0, **1**, 2, 3, 4, 5, 6, 7, **8**, 9)	→	(0, **8**, 2, 3, 4, 5, 6, 7, 1, 9)
(0, 1, **2**, 3, 4, **5**, 6, 7, 8, 9)	→	(0, 1, **5**, 3, 4, **2**, 6, 7, 8, 9)
(A, B, **C**, D, E, F, **G**, H, I, J)	→	(A, B, **G**, D, E, F, **C**, H, I, J)
(A, **B**, C, D, E, F, G, H, I, **J**)	→	(A, **J**, C, D, E, F, G, H, I, **B**)

Tab. 19 Jak je vidět, tento operátor je možné použít u všech typů kódování (binární, reálné, celočíselné i permutační).

d) Posuvná mutace

V chromozómu je náhodně zvolen gen a ten je posunut na náhodně vygenerovanou pozici, tímto se nám posunou pozice genů, odkud vkládáme do místa, kde bereme. *Tab. 20* nám reprezentuje tento typ mutace. Jak je vidět, tento operátor je možné použít pro všechny typy kódování. [11]

Reprezentace kódování	Původní chromozóm		Zmutovaný chromozóm
Binární kódování	(0, 1, 1, 0, 0, 1, 1, 0, **0**, 0)	→	(0, 1, 1, 0, 0, **0**, *1*, *1*, *0*, 0)
Permutační kódování	(A, B, C, D, E, F, **G**, H, I, J)	→	(A, B, **G**, *C*, *D*, *E*, *F*, *H*, I, J)
Reálné kódování	(1.22, 8.11, **3.44**, 1.11)	→	(1.22, **3.44**, *8.11*, 1.11)

Tab. 20 Posuvná mutace.

e) Inverzní mutace

V *Tab. 21* je ukázáno, jak operátor pracuje. Nejdříve jsou zvoleny náhodně dva geny v chromozómu a následně dojde k invertování úseku mezi těmito geny. Tento operátor můžeme tedy použít i pro permutační kódování. [23]

Reprezentace kódování	Chromozómy před mutací		Chromozómy po mutaci
Binární kódování	(1, **0**, **0**, **1**, **0**, **0**, **0**, 0, 1)	→	(1, **0**, **0**, **0**, **1**, **0**, **0**, 0, 1)
Celočíselné kódování	(3, 2, 1, **7, 6**, 5, 9, 8, 0)	→	(3, 2, 1, **6,** **7**, 5, 9, 8, 0)
Permutační kódování	(**A**, **B**, **C**, D, E, F, G, H, I)	→	(**C**, **B**, **A**, D, E, F, G, H, I)

Tab. 21 Inverzní mutace.

6.7 Doplnění populace

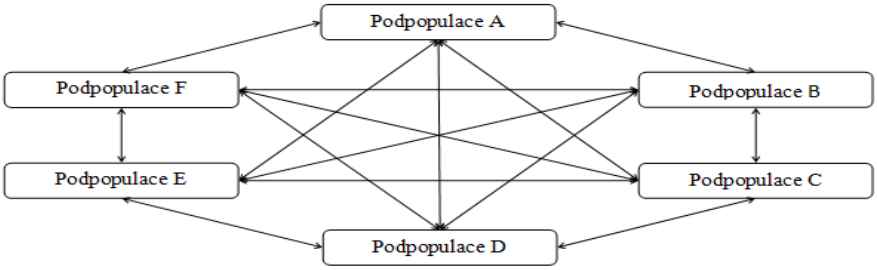
Tento operátor přichází na řadu po selekci, křížení a mutaci. Pokud je vyprodukováno méně nebo více jedinců než v původní populaci, je potřeba tuto skutečnost napravit. K tomuto účelu se používá více přístupů, které je taktéž možné najít ve zdroji [23].

6.8 Paralelní genetické algoritmy

Tato podoba Genetických algoritmů byla navržena pro urychlení výpočtu hledaných řešení. K dispozici máme tři hlavní modely paralelních Genetických algoritmů, a to Migrační, Globální a Difuzní model.

a) Migrační model

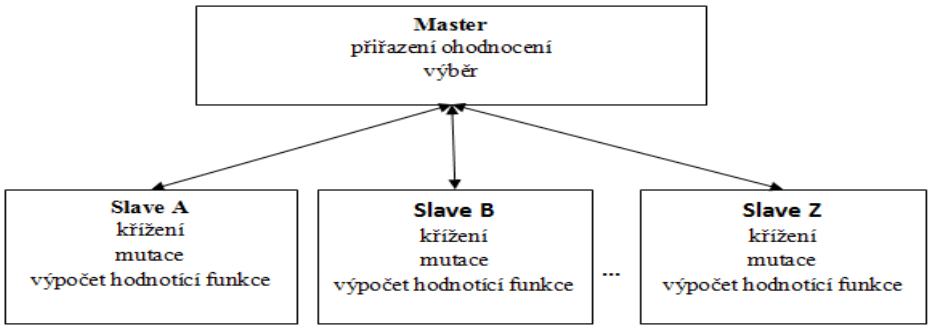
Populace je rozdělena do více podpopulací *Obr. 10*. Tyto podpopulace se vyvíjejí nezávisle na ostatních po stanovený počet iterací. Po výpočtu všech iterací dochází k migraci určitého počtu jedinců z každé podpopulace do jiné podpopulace. Na základě druhu selekce jedinců pro výměnu a způsob výměny jedinců určuje rozmanitost a změnu informací, která může nastat mezi jednotlivými podpopulacemi. Na tomto základě dělíme migrační modely, to je možné najít v [23]. Implementace migračního modelu nám zajistí rychlejší dosažení optima při méně hodnotících funkcích ve srovnání s pouze jednou populací. [23]



Obr. 10 Migrační model

b) Globální model

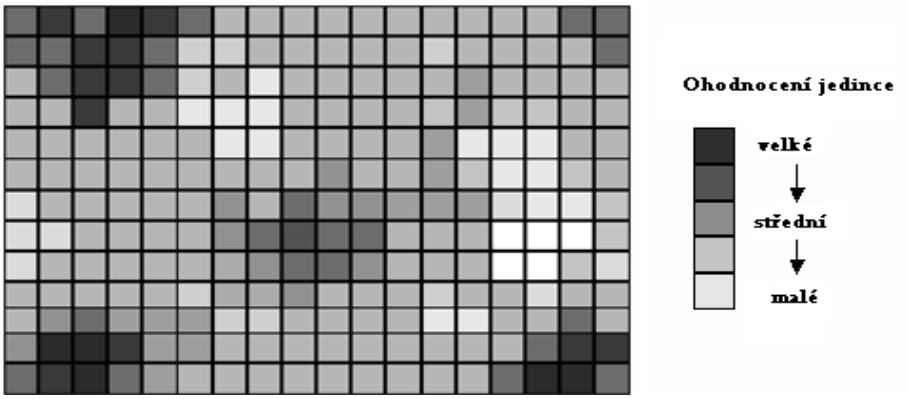
Tento model nerozděluje populace na více podpopulací, ale rozděluje průběh výpočtu Genetického algoritmu do paralelních podprocesů. Řídící proces (master) provádí selekci a přiřazuje ohodnocení jedincům, ostatní procesy jako je křížení, mutace a výpočet fitness funkce jsou distribuovány na podřízené procesy (slave) *Obr. 11*. [23]



Obr. 11 Globální model.

c) Difuzní model

Každý jedinec je řízen zvlášť a jedinci vstupují do křížení s jinými jedinci pouze v jejich okolí. Jak je patrné z *Obr. 12.* [23]



Obr. 12Difuzní model populace.

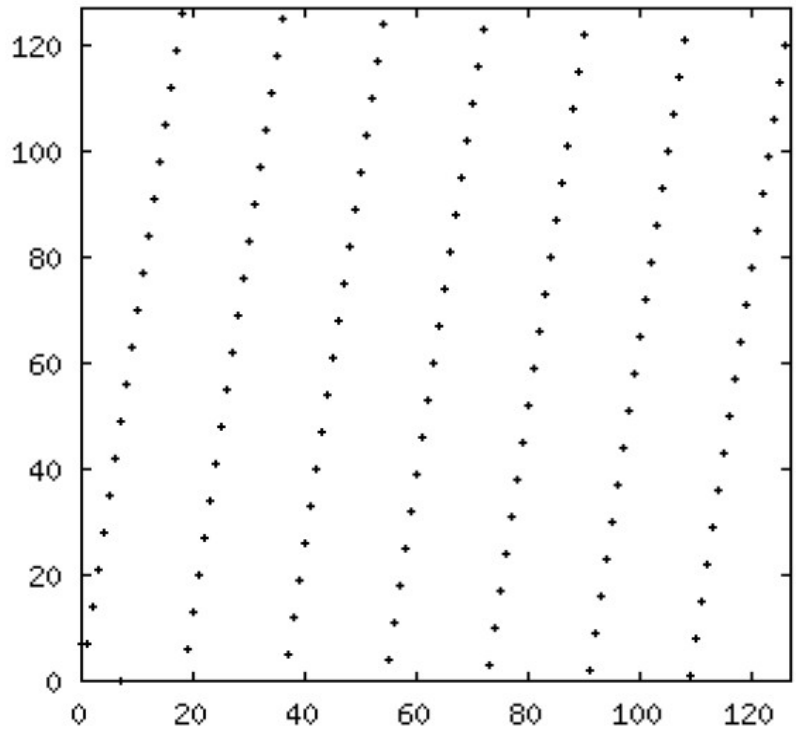
7 GENEROVÁNÍ PSEUDONÁHODNÝCH ČÍSEL

Generování pseudonáhodných čísel je efektivní deterministický algoritmus, který generuje posloupnost čísel, které by měly být statistickými testy nerozlišitelné od náhodných. Existují skutečně generátory náhodných čísel, jako jsou kvantové generátory, šum, nicméně ne vždy jsou takové zařízení založené na skutečných fyzikálních pochodech k dispozici. Generátory pseudonáhodných čísel mají široké využití v kryptografii, počítačových hrách, numerických metodách, apod. Jako vstupní data pro generátory pseudonáhodných čísel slouží krátké posloupnosti skutečně náhodných čísel tzv. random seed („náhodné semínko“). Vzhledem k tomu, že generátory obsahují deterministické algoritmy a počítače disponují ohraničenou pamětí, začnou být sekvence, které tyto generátory produkují časem nutně periodické. “*Random number generators should not be built at random*” [17].

Skutečně náhodná posloupnost délky *n* je taková posloupnost, k jejímuž vyjádření potřebujeme všech *n* čísel, není ji tedy možné vyjádřit v nějakém explicitním tvaru pomocí parametrů a podmínek. Většina dostupných generátorů pseudonáhodných čísel generuje posloupnost s rovnoměrným rozložením. [18]

7.1 Lineární kongruentní generátory

Čísla jsou generována podle vztahu $x_{n+1} = (a\ x_n + c)\ mod\ m$. Kde *mod* znamená funkci modulo, tedy zbytek po celočíselném dělení. Parametry *a, c, m* jsou přirozené konstanty. Na jejich volbě závisí výsledná kvalita generátoru. Získaná řada pseudonáhodných čísel je velikosti od *0* do *m - 1*. Tato řada je periodická s periodou nejvýše *m*. Na *Obr. 13* je možné vidět rozložení vygenerovaných bodů do plochy $[x_n; x_{n+1}]$. Jak je vidět, plocha není vyplněna zdaleka hustě a úplně rovnoměrně. Tyto typy generátorů je možné dělit na lineární (výše uvedený), multiplikativní $(x_{n+1} = (a\ x_n)\ mod\ m)$ a aditivní $(x_{n+1} = (x_n + x_{n-1})\ mod\ m)$. Liší se tedy podle toho, jaký vztah používají ke generování čísel. [18]



Obr. 13 Body pro $x_{n+1}\ (7\ x_n + 0)\ mod\ 127$ [18]

Požadavky na generátory pseudonáhodných čísel:

- uniformita
- nezávislost
- dlouhá perioda
- jednoduchost implementace
- rychlost výpočtu

V *Tab. 22* je možné vidět hodnoty, které by generoval lineární generátor náhodných čísel s následujícím předpisem $x_{n+1} = (5x_n + 3) \bmod 16$, $x_0 = 7$. K ohodnocení kvality generátorů náhodných čísel slouží celá řada testů, jako jsou statistické (*spektrální analýza , K-S, $\chi 2$*), které se řadí mezi empirické testy nebo strukturální, které jsou pro teoretické prověření získaných hodnot. Všechny tyto metody nám sice neříkají nic o tom, že by generátory byly opravdu skutečně generátory náhodných čísel, ale pomáhají nám ověřit míru jejich náhodnosti. [2]

n	Xn	Yn	n	Xn	Yn	n	Xn	Yn	n	Xn	Yn
0	7	---	5	10	0.625	10	9	0.563	15	4	0.250
1	6	0.375	6	5	0.313	11	0	0.000	16	7	0.438
2	1	0.063	7	12	0.750	12	3	0.188	17	6	0.375
3	8	0.500	8	15	0.938	13	2	0.125	18	1	0.063
4	11	0.688	9	14	0.875	14	13	0.813	19	8	0.500

Tab. 22 Sekvence pseudonáhodných čísel [2]

7.2 Některé implementace generátorů

a) Systémový generátor

Systém Windows má v sobě zabudovaný vestavěný generátor v knihovně System.Random, který je založen na algoritmu z publikace [17]. Z několika hledisek není tento generátor ideální, protože v některých testech nedosahuje ideálních hodnot.

b) Ranlux generátor

Tato sada generátorů náhodných čísel byla vyvinuta v roce 1994 M. Luescherem. Pseudonáhodná čísla jsou generována v 24 bit posloupnosti. Je zde zabudovaný mechanismus pro snížení korelace mezi jednotlivými hodnotami. Generátory jsou rozděleny do několika úrovní a podle toho se liší kvalita generované sekvence čísel a rychlost jejího vytváření.

c) Zobecněný zpětnovazebný posuvný registr (GFSR - Generalized Feedback Shift Register)

Generátor kombinuje 4 hodnoty z dlouhé sekvence čísel v registrech a vytváří tak nové, náhodné 32 bitové číslo. Nevýhodou tohoto generátoru je dlouhá doba inicializace generátoru, ale samotná generace náhodných čísel je rychlá.

d) Mersenne Twister generátor

Tento generátor vyvinul v roce 1997 Makoto Matsumoto a Takuji Nishimura. Tento generátor má oproti ostatním dlouhou periodu 2^{19937}-1. Umožňuje rychlé vytváření vysoce kvalitních pseudonáhodných čísel. Existují dvě varianty tohoto algoritmu a to MT19937 s 32-bit délkou slova a 64-bit, MT19937-64.

Tab. 23 udává rychlost produkce pseudonáhodných čísel za sekundu. Testovací PC bylo Pentium IV osazené 3GHz procesorem, výsledné hodnoty jsou v milionech za sekundu.

[19]

Generator	Integers/s	Doubles/s
System.Random	26,9	44,5
MersenneTwister	36,7	13,0
RanLux (default)	7,6	8,0
RanLux (better)	5,0	5,2
RanLux (best)	2,8	2,9
GsfrGenerator	49,6	16,3

Tab. 23 Počet milionů vygenerovaných pseudonáhodných čísel za sekundu [19].

Tab. 24 reprezentuje test mnoha různých generátorů pseudonáhodných čísel. K výpočtu byla použita Caesarova sumace [21], což je matematická metoda pro výpočet nekonečných řad, zde byla použita pro výpočet hodnoty Pi, bylo uděláno 1500 experimetů pomocí metody Monte-Carlo, která slouží pro simulaci systémů v informatice. Typicky se používá pro výpočet integrálů nebo diferenciálních rovnic. Více o metodě v [20].

Mersenne Twister (MT19937) má nejlepší výsledky v porovnání s ostatními generátory, hodnota Pi je nejpřesnější. Například systémový generátor RAND-ANSI, je oproti Mersenne Twisteru slabý. [22]

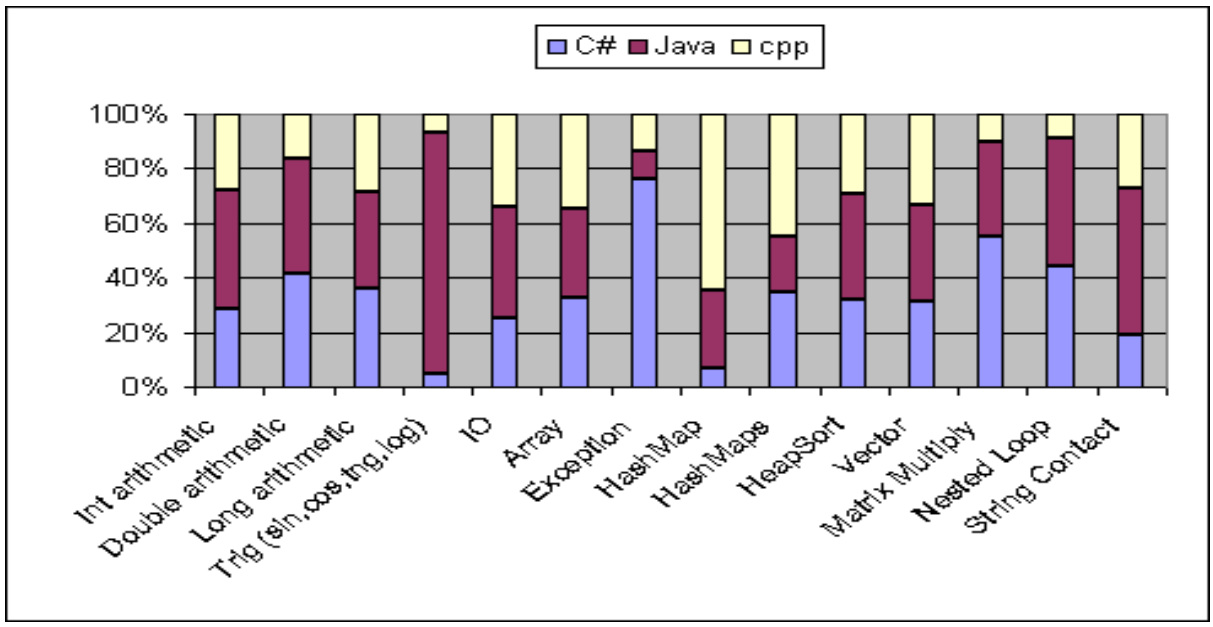
Generator	Estimated Pi	Error	Accuracy(%)
MT19937	3.14174882107861	-0.000156167	99.99502903
MT11213A	3.14312799095236	-0.001535337	99.95112869
MT11213B	3.14641081249572	-0.004818159	99.84663324
TT800	3.15771979876276	-0.016127145	99.48665703
T400	3.13436676124481	0.007225892	99.76999270
T403	3.15929528896189	-0.017702635	99.43650761
T775	3.13009871443637	0.011493939	99.63413655
T800	3.13402467483056	0.007567979	99.75910375
TAUSWORTHE7-3	3.23104072607105	-0.089448072	97.15277942
RANDOM-SCHEME	3.14416356112738	-0.002570908	99.91816547
RAND-ANSI	3.14347306730965	-0.001880414	99.94014457
RANDU	2.72479676423189	0.416795889	86.73297479
LM	3.15057227466379	-0.008979621	99.71416978
MINSTD1	3.13060995963610	0.010982694	99.65040999
MINSTD2	3.14019942295212	0.001393231	99.95565209
SUPER-DUPER	2.71774363918035	0.423849014	86.50846685
BCPL	2.72817475234362	0.413417901	86.84049949
LCG-SICP	3.38448721711206	-0.242894564	92.26842591
LCG-276	3.58568582800318	-0.444093174	85.86407522
LCG-277	3.16227766016837	-0.020685007	99.34157579

Tab. 24 Porovnání jednotlivých generátorů pro výpočet hodnoty Pi [22].

8 PRAKTICKÁ ČÁST

Byla navržena a implementována aplikace pro účely plánování výroby, popsané v kapitole 2. Aplikace byla vytvořena ve třech vrstvách. Aplikační část, která se stará o výpočty, byla vytvořena jako knihovna v jayzce C++ v nativním, unmanaged kódu [12], to nám zajistilo kratší dobu výpočtu. Na *Obr. 14* je možné vidět srovnání jednotlivých jazyků a jejich rychlostí při výpočtech. Čím je sloupec delší, tím delší čas u daného jazyka test zabral. Pokud vezmeme v potaz celkový čas všech testů a uděláme srovnání, pak C# zabral 38,27 %, Java 41,61 % a C++ 20,12%. Jak je vidět, C++ je mnohem rychlejší. [15]

Více informací o managed a unmanaged kódu je možné najít zde [14]. Prezentační část byla vytvořena v C# pod .NET frameworkem samozřejmě už ve spravovaném, managed kódu a datová část je reprezentována soubory XML, které se ukládají do adresáře samotné aplikace. Pro práci s daty a těmito soubory byla použita technologie LINQ do XML taktéž v rámci .NET platformy [13]. Komunikace mezi knihovnou C++ a prezentačním prostředím v C# je zajištěna pomocí knihovny napsané v C++ managed kódu tak, aby bylo možné pohodlně spouštět rutiny pro výpočty přímo z prostředí C# kódu.



Obr. 14 Srovnání programovacích jazyků a jejich rychlosti [15].

Následující část této kapitoly bude prezentovat výsledky jednotlivých heuristických metod aplikovaných na daný problém v plastikářské výrobě, který byl rozebrán v kapitole 2. Všechny sestavené plány a jejich výsledky, které byly v práci využity, je možné najít v adresáři aplikace, ve formátu XML nebo TXT, která je na CD přiložena k práci.

8.1 Konkrétní specifikace řešeného problému

Než mohla být aplikace vytvořena, bylo potřeba přesně určit návrh aplikace. Pro sestavování plánů a následující fitness funkci byly vybrány tyto nejdůležitější podmínky:

- Priority zakázek
- Vhodnost výrobku na stroj
- Po sobě jdoucí výrobky, které se liší barvou, ale vstřikovací formu mají stejnou
- Deadline - čas ukončení
- Materiál - jeho přesný název
- Typ materiálu
- Barva materiálu
- Obsluha

Jedinec byl reprezentován posloupností čísel, např. (4, 65, 33, 8, …, 22). Prvky z této posloupnosti označují identifikační čísla jednotlivých úkolů zadaných pro konkrétní plán. Na základě této posloupnosti algoritmus sestaví plán následujícím způsobem:

1. Inicializuj výrobní seznamy pro 13 strojů
2. vezmi první dosud nezařazený úkol z posloupnosti úkolů
3. zjisti, na které stroje může být přiřazen
4. přiřaď úkol na stroj, který je v čase nejdřívě dostupný
5. opakuj od bodu 2., dokud nebudou všechny úkoly přiřazeny na stroje

Takto sestavený plán je možné ohodnotit dle příslušných kritérií a sestavit tak fitness funkci. Tato kritéria vyplynula ze zkušeností zaměstnanců daného výrobního závodu. Než se však podíváme na způsob realizace této funkce, zastavíme se nejdříve u výběru metod, které bylo možné u tohoto problému použít.

Jak již bylo několikrát zmíněno, jedná se o problém obsahující 13 nezávislých strojů, na které jsou umisťovány jednotlivé úkoly. Pro představu stavového prostoru je důležité vědět, že se bude jednat o rozvržení asi 100 až 200 prací na jeden měsíc. Stroje se liší svou velikostí, ale i kvalitou a rychlostí. Dle vstřikovací formy pro každý výrobek může být úloha přiřazena jen určitým strojům a na tomto základě takový úkol získá svůj výrobní čas. Z tohoto hlediska je možné uvažovat o modifikaci přesného algoritmu z kapitoly 4.3. d) Nezávislé paralelní stroje, ale jak je i tam uvedeno, již pro 5 strojů a 50 úkolů, při použití osekání stavového prostoru pomocí metody Větví a mezí se jedná o *NP úplný* problém.

Můžeme se na problém podívat z jiného pohledu a pokusit se o výpočet všech možností hrubou silou. V našem případě se jedná o posloupnost úkolů, kde záleží na pořadí, protože na tomto základě je plán sestavován a žádný z úkolů se v této posloupnosti nesmí opakovat. Uvažujme nejjednodušší případ, a to 100 úkolů v plánu. Pak se jedná o výpočet všech permutací bez opakování, a to je 100!. *Tab. 25* nám přehledně ukazuje počet možností a dobu výpočtu těchto možností. Jak je vidět výpočet všech řešení nepřipadá v úvahu.

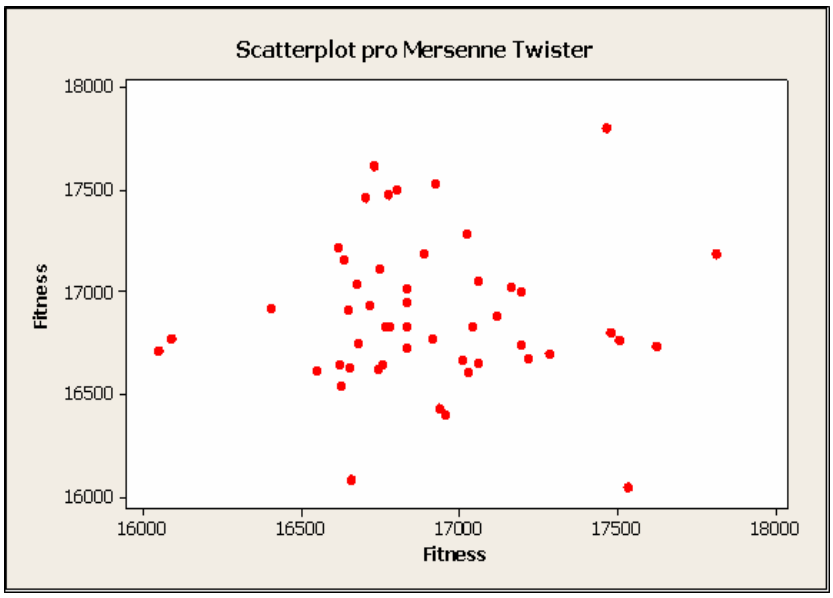
Počet operací [-]	Počet permutací [-]	Počet možností [-]	Délka výpočtu pro jeden plán [ms]	Celková doba výpočtu všech řešení [roky]
100	100!	9.3344E+157	26	2.94E+146

Tab. 25 Použití hrubé síly.

Z tohoto důvodu bylo přistoupeno k heuristickým metodám. Vzhledem k tomu, že byly použity stochastické heuristické metody, kde se využívá generování náhodných čísel, byly použity dva vybrané z kapitoly 6. a to standardní vestavěný v System.Random a Mersenne Twister generátor pseudonáhodných čísel.

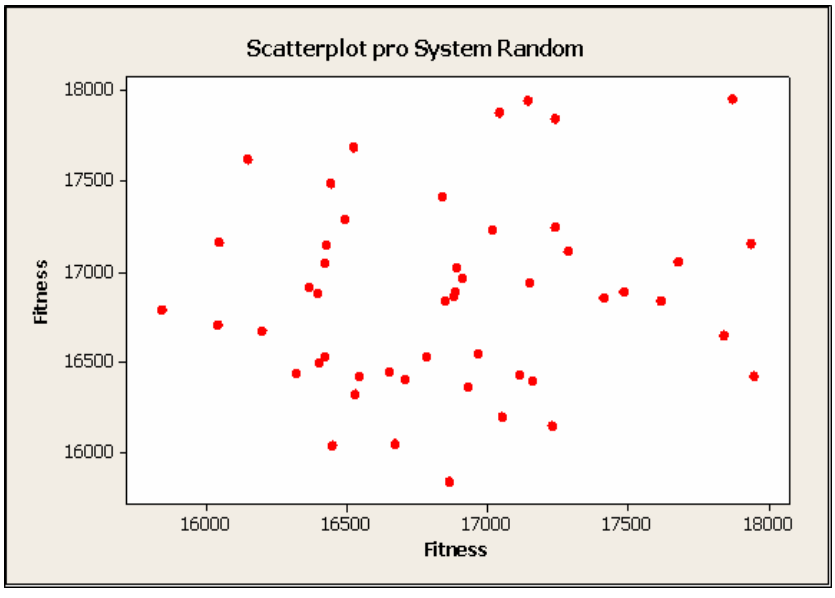
8.2 Srovnání vybraných random generátorů

Než se pustíme do srovnání obou generátorů, je potřeba uvést, že testy v této kapitole nereprezentují náhodnost generovaných čísel, ale náhodnost generovaných rozvrhů při stejných počátečních podmínkách. Z pohledu softwaru nám půjde o to tuto náhodnost co nejvíce redukovat tak, aby uživatel při použití tohoto softwaru dostával věrohodné či podobné rozvrhy pro stejný test při stejných výchozích datech. Půjde nám tedy o robustní chování aplikace. Záměrně byla pro účely srovnání nastavena populace na 10 jedinců, a ta byla 5x iterována. Takto krátký test ukázal, který z random generátorů lépe vyhovuje při užití metod Genetických algoritmů.



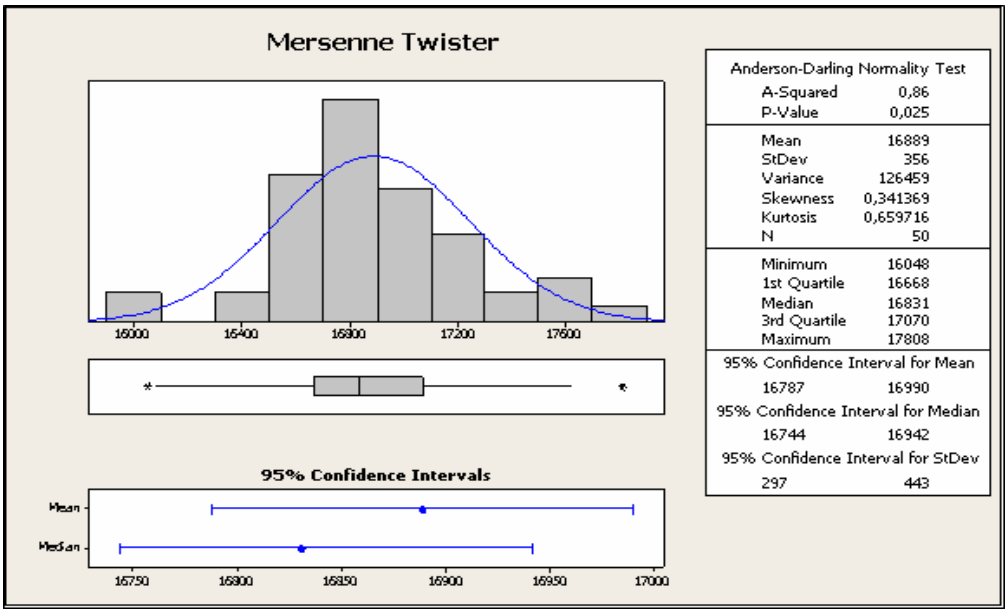
Obr. 15 Scatterplot pro analýzu robustnosti programu při použití Mersenne Twister generátoru.

Pro každý generátor bylo sestaveno 50 plánů za stejných výchozích podmínek. Bylo tedy pokaždé vybráno počáteční sestavení obsahující stejný počet identických úkolů. Každý plán si nese své ohodnocení v podobě fitness funkce. Na tomto základě můžeme srovnat použití obou random generátorů. Soubor 50 hodnot je v našem případě dostačující ukazatel pro statistické srovnávání. K tomuto účelu byl použit software pro statistické výpočty Minitab [16].



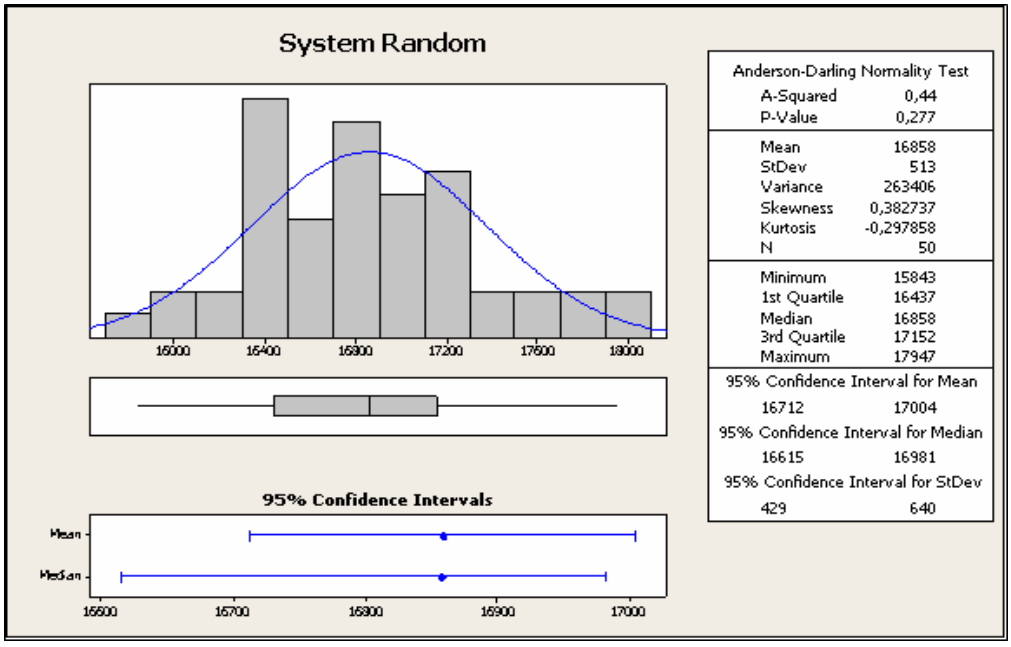
Obr. 16 Scatterplot pro analýzu robustnosti programu při použití System Random generátoru.

Z Obr. 15 a 16 je patrné, že při použití Mersenne Twister generátoru se hodnoty více soustředí do stejného rozmezí někde mezi fitness funkce rovné 16600 až 17300. Oproti tomu při použití systémového generátoru náhodných čísel se výsledky fitness funkce pohybují rovnoměrně v rozmezí od 16000 až po 18000. Jak je tedy vidět, použití Mersene Twister genrátoru nám zajišťuje mnohem robustnější chování aplikace.



Obr. 17 Souhrnný statistický test na Normálním rozdělení při použití Mersenne Twister.

Samozřejmě z pohledu uživatele takového softwaru je výhodnější, aby měl jistotu, že když plánování spustí, dostane pokaždé podobnou hodnotu. Není přípustné, aby jednou dostal plán, který bude mít hodně špatné ohodnocení, a jednou ohodnocení zase dobré. Tedy jde nám o to, aby výsledky plánování nebyly náhodné rozvrhy, ale aby v obdržených datech byla jistá závislost, v našem případě pohyb výsledné fitness hodnoty v co nejužším intervalu. Z tohoto pohledu vychází Mersenne Twister generátor pseudonáhodných čísel mnohem lépe a z toho důvodu byl i v této práci použit a do softwaru zakomponován.



Obr. 18 Souhrnný statistický test na Normálním rozdělení při použití System Random.

Jak je patrné z *Obr. 17 a 18* při použití Mersenne Twisteru generátoru mají hodnoty fitness opravdu užší rozptyl, to můžeme vidět na Gaussově (Normálním) pravděpodobnostním rozdělení nebo taktéž vyčíst z ostatních hodnot, které byly obdrženy ze statistických testů. V našem případě je *StDev* směrodatná odchylka, která nám říká, jak moc jsou hodnoty rozptýleny od střední hodnoty - v našem případě se jedná o hodnotu *Mean*. Taktéž si můžeme všimnout, že maximální a minimální hodnota má při použití systémového generátoru mnohem větší rozptyl. Tuto tendenci nám potvrzuje i medián, tedy hodnota, která se nachází uprostřed seznamu. Při použití systémového generátoru je tato hodnota dokonce totožná s průměrem, což značí podobnou frekvenci výskytu obdržených hodnot v celém rozsahu.

8.3 Fitness funkce

Výčet jednotlivých podmínek z kapitoly 7.1. nám pomůže při určování fitness funkce. Kvalita každého rozvrhu je jednoznačně dána mírou splnění těchto podmínek (priorita zakázky, vhodnost umístění na stroj, materiál, deadline, barva, apod.) . Na základě konzultace ve výrobním závodu bylo zjištěno, že jednotlivé podmínky mají odlišnou váhu, která se z větší části měří finančními ztrátami. Podmínka priority zakázky udává, pro koho bude výrobek vyroben. Například vyrobit výrobek přednostně pro dobře platící automobilku, která odebírá ve velkém množství, má mnohem větší váhu než správné následování barev dle barevné škály. Takto vytvořená fitness funkce nám bude vzniklé řešení posílat do určité oblasti stavového prostoru, protože bude zvýhodňovat specificky sestavené plány. Největší váhy tedy byly nastaveny na upřednostnění prioritních zakázek, dále to, jestli je forma na správném stroji, a po sobě jdoucí výrobky lisované v jedné formě lišící se barvou nebo materiálem. Následovalo hodnocení data ukončení před datem odevzdání a strojní obsluha. Na konci vah hodnocení byl název materiálu nebo typ a po sobě jdoucí barevná škála.

```
//deadline
timeInD = timeInD + q->data->TimeInMinutes();
if(q->data->DeadLineFromBeginingInMinutes() > timeInD)
{
    help = help + 25;//50
}
//práce na stejné formě
if(q->link->data->NumberOfplot() == q->data->NumberOfplot())
{
    help = help + 75; //100
}
// po sobě jdoucí barvy
int minusOperation =  q->link->data->ColourOfMaterial() - q->data->ColourOfMaterial();
help = help + minusOperation;
if(minusOperation > 0)
{
    help = help +10/minusOperation; //10
}
//stejný materiál
if(q->link->data->NameOfMaterial() == q->data->NameOfMaterial())
{
    help = help +15; //20
}
//stejný typ materiálu
if(q->link->data->TypeOfMaterial() == q->data->TypeOfMaterial())
{
    help = help +12;
}
j++;
```

Obr. 19 Část kódu hodnotící funkce a nastavení vah pro některé podmínky

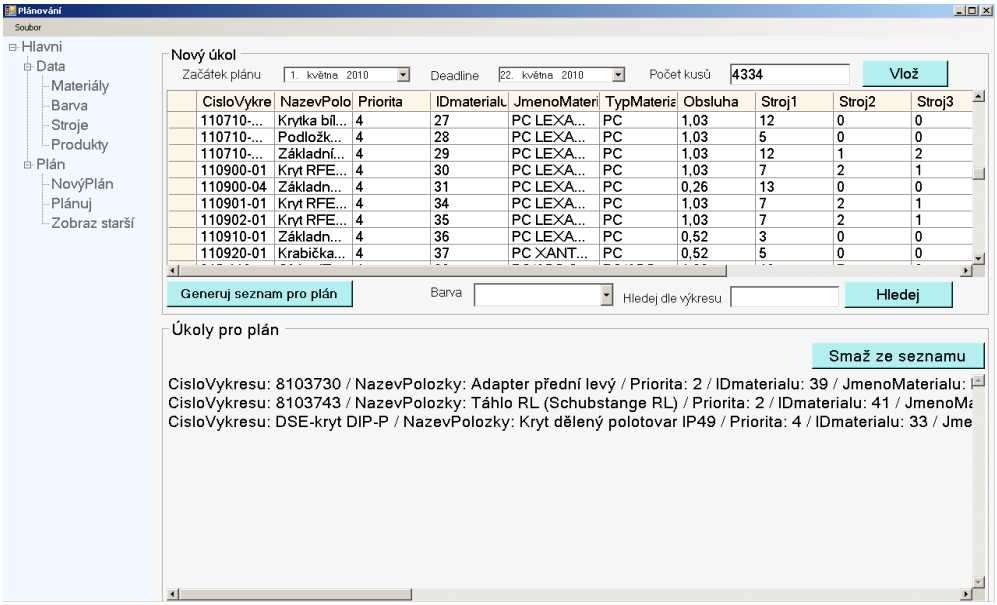
Na Obr. 19, je možné vidět část hodnotící funkce a nastavení hodnot pro některé podmínky. Fitness funkce je tedy vypočtena tak, že každý plán začíná s nulovým ohodnocením a postupně k tomuto hodnocení jsou nebo nejsou přičítany číselné hodnoty. Pokud je například dodržen čas dokončení (deadline) je k fitness funkci přičtena hodnota 50, při splnění podmínky po sobě jdoucích stejných forem je přičtena hodnota 75, apod. V podobném duchu byly oceněny další požadované podmínky, jak je možné vidět například na *Obr. 20.*

```
int placed = priority[0];
int prMach = 4;
double timeHoleBig = listOfMachines[priority[0]-1]->CountHoleTime();

for(int j = 1; j < 4; j++)
{
    if(priority[j] != 0)
    {
        if(timeHoleBig > listOfMachines[priority[j]-1]->CountHoleTime())
        {
            placed = priority[j];
            prMach = 4 - j;
        }
        else
        {
            timeHoleBig = listOfMachines[priority[j]-1]->CountHoleTime();
        }
    }
}
listOfMachines[placed-1]->Append(job);

// ocenění priority stroje pro konkrétní úkol, např: 4*10 = 40 nebo 3*10 = 30 apod.
fitness = fitness + prMach * 10;
```

Obr. 20 Další část kódu pro ohodnocení priority stroje pro konkrétní úkol

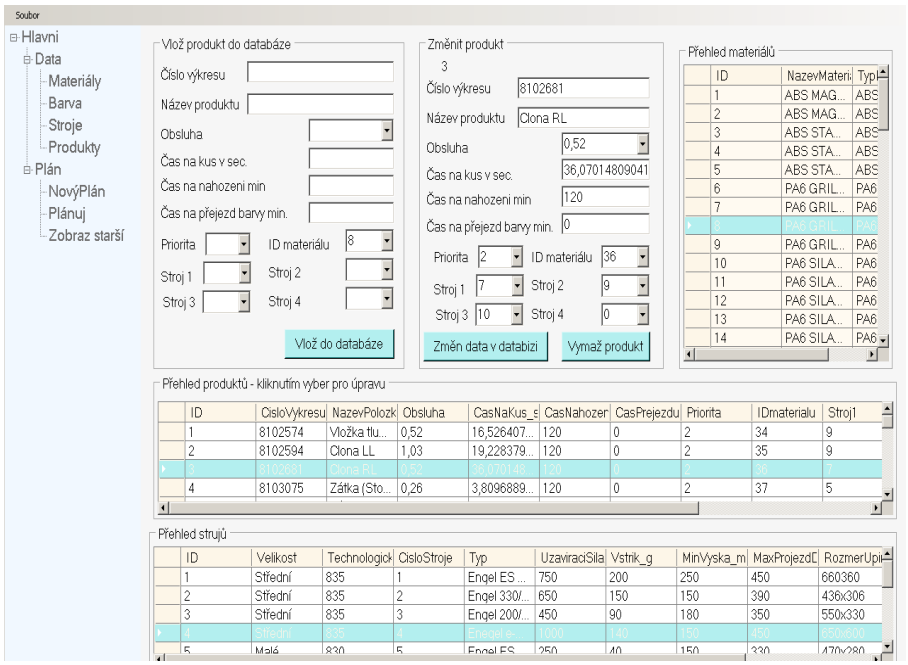
8.4 Představení aplikace a srovnání heuristických metod

Výsledné plány byly sestavovány za použití výše uvedené fitness funkce. Pro Genetický algoritmus byly vybrány následující metody, a to Turnajový výběr, Křížení s částečným překřížením a Posuvná mutace, do algoritmu byl zakomponován i elitismus. Při této kombinaci dosahoval při použití stejných podmínek Genetický algoritmus v aplikaci na tento konkrétní případ rozvrhování plastikářské výroby nejlepších výsledků. Pro srovnání byly implementovány další heuristické metody, a to Horolezecký algoritmus, Zakázané prohledávání a Simulované žíhání. Všechny uvedené metody jsou popsány v kapitole 5. a 6.

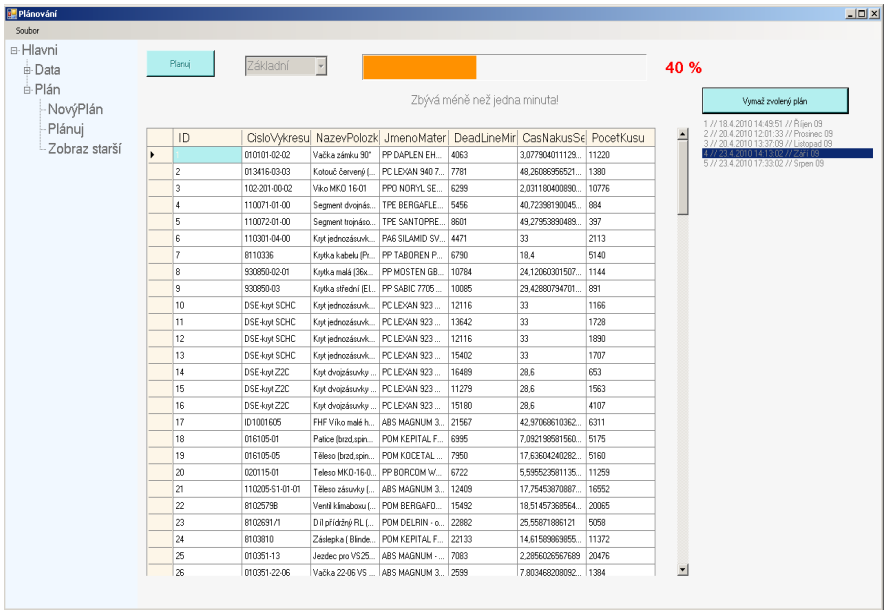


Obr. 21 Zadávání a vytváření nového plánu.

Sestavené rozvrhy je možné porovnat s plány, které byly v daném závodě sestaveny ručně. Vytvářením rozvrhů se zde zabývají dva odborníci, pan Ing. Hlavinka a pan Ing. Slováček. Pro závod je důležité mít sestavené rozvrhy co nejvíce efektivně, protože na tom může ušetřit spoustu peněz. Plány jsou zde sestavovány ručně, ale závod má s touto činností dlouholeté zkušenosti. K dispozici byl celý rok 2009, pro účely naší aplikace bylo do softwaru zadáno 5 rozvrhů a ty následně srovnány s výsledky získanými na základě heuristických metod. Aplikace byla navržena jednoduše tak, aby do ní bylo možné vkládat nová data a sestavovat budoucí rozvrhy. To je možné vidět na *Obr. 21, 22 a 23.*

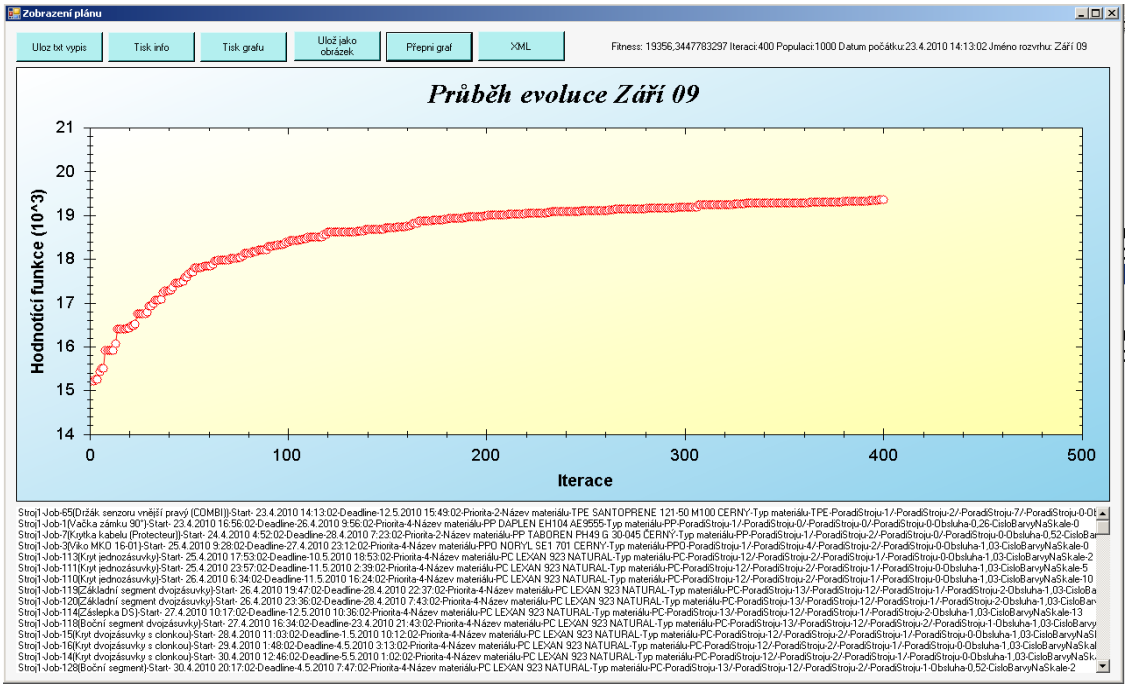


Obr. 22 Zadávání, úprava nebo mazání výrobku v seznamu výrobků.



Obr. 23 Hledání optima pro vytvořený plán.

V aplikaci je možné se k jednotlivým výsledkům plánování vrátit a získané rozvržení pro jednotlivé stroje exportovat do textového, XML nebo grafického souboru, jak je možné vidět na *Obr. 24.*



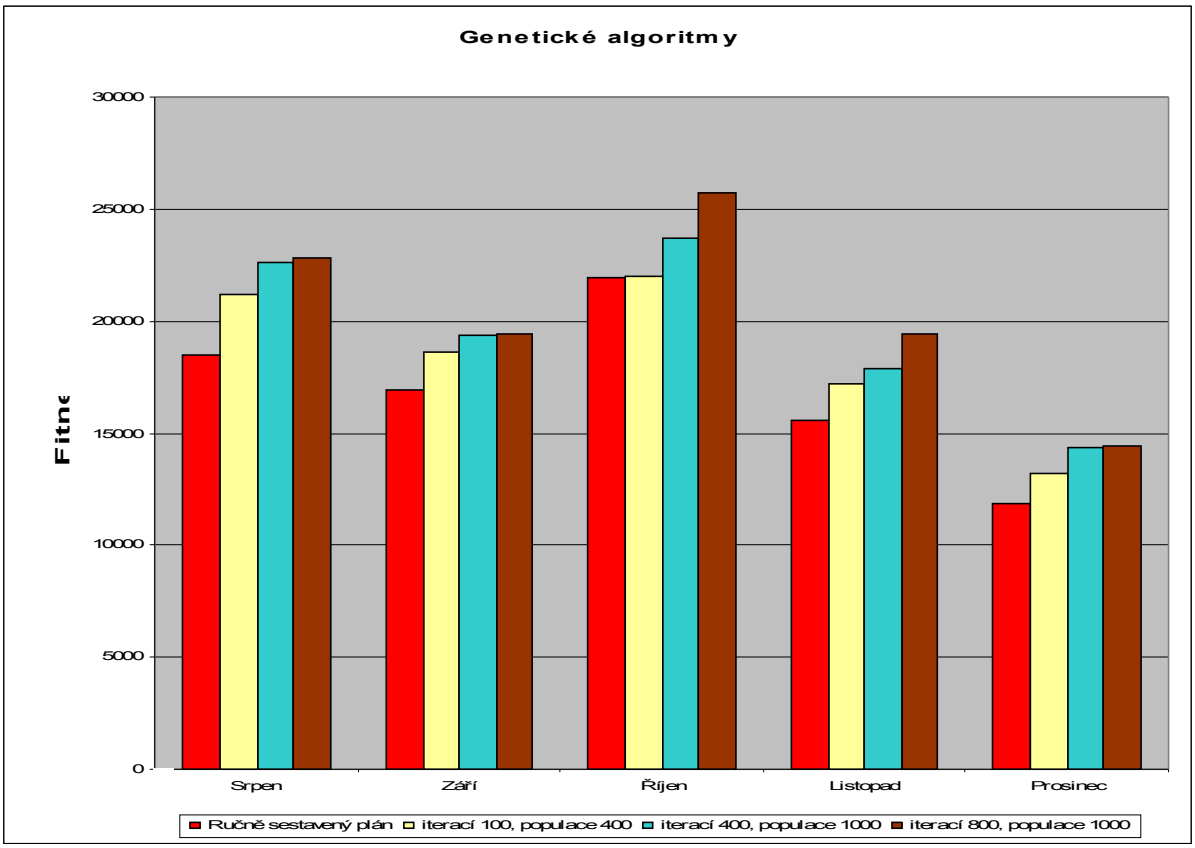
Obr. 24 Výsledek po optimalizaci.

ručně sestavený plán [-]	Iterací 100, populace 400 [-]	nárust kvality [%]	Čas [min]	Iterací 400, populace 1000 [-]	nárust kvality [%]	Čas [min]	Iterací 800, populace 1000 [-]	nárust kvality [%]	Čas [min]	Počet úkonů
18474	21186	14,7	5	22652	22,6	60	22850	23,7	91	143
16950	18650	10,0	5	19356	14,2	55	19455	14,8	103	145
21937	21986	0,2	5	23684	8,0	63	25737	17,3	98	154
15564	17195	10,5	4	17871	14,8	52	19460	25,0	86	137
11821	13237	12,0	3	14373	21,6	35	14421	22,0	52	96
16949	18451	9	4	19587	16	53	20385	21	86	135
ručně sestavený plán [-]	Horolezecký algoritmus okoli 1000 [-]	nárust kvality [%]	Počet iterací [-]	Zakázané prohledávání Iterací 400, okolí 1000, zákaz 25 [-]	nárust kvality [%]	Čas [min]	Simulované žíhání /Teplota 50 - 1500/kmax 1000/alfa 0.99 [-]	nárust kvality [%]	Čas [min]	Počet úkonů
18474	21929	18,7	117	21344	15,5	21	20488	10,9	20	143
16950	18462	8,9	78	19415	14,5	21	16918	-0,2	21	145
21937	23340	6,4	91	23973	9,3	23	22666	3,3	22	154
15564	17197	10,5	75	18171	16,8	19	18158	16,7	20	137
11821	13567	14,8	54	14149	19,7	14	12422	14,0	14	96
16949	18899	12	83	19410	15	20	18130	9	19	135

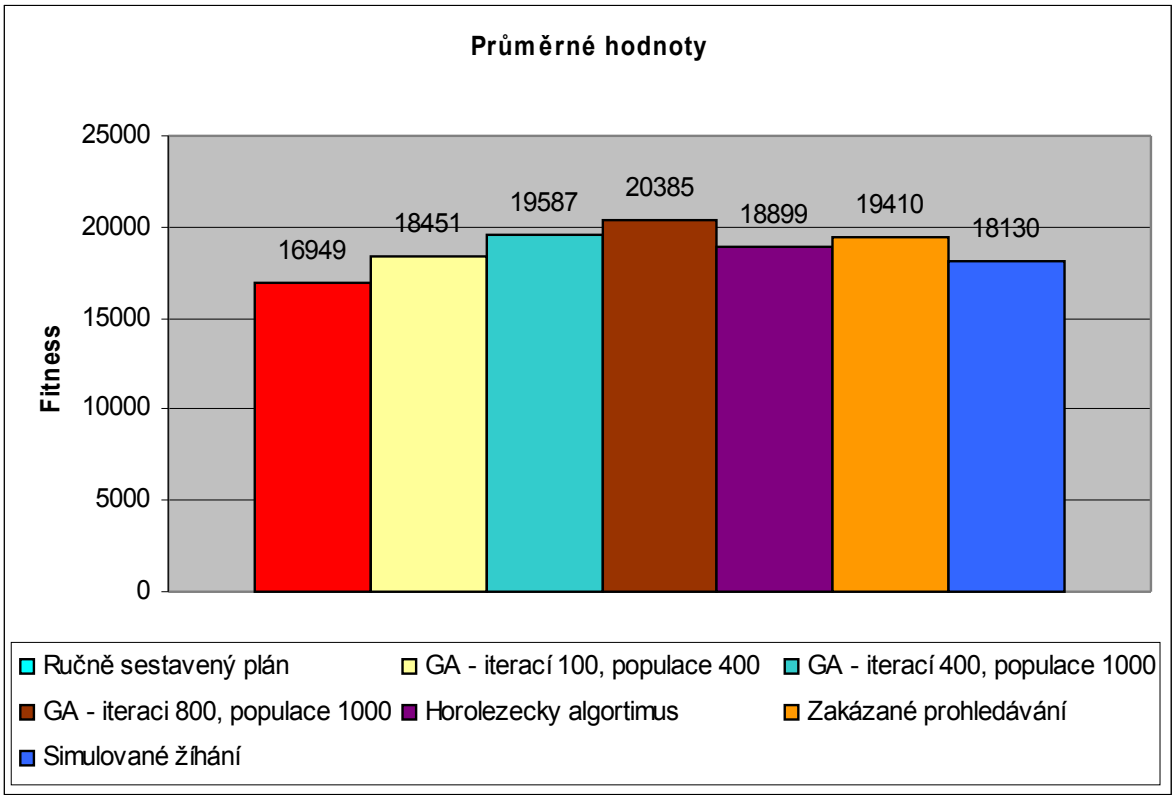
Tab. 26 Výsledky plánování při použití heuristických metod v závislosti na fitness funkci

Jak je patrné z *Tab. 26* a ostatně i z *Obr. 25, 26 a 27,* dosahují Genetické algoritmy v porovnání se základními heuristikami lepších výsledků při srovnatelných dobách výpočtu. Mluvíme zde o plánu pro 100 až 200 úkolů rozvržených na 13 strojů, kde muselo být splněno spoustu podmínek. Z tohoto důvodu si můžeme všimnout, že nárůst kvality řešení sestaveného počítačem je oproti ručně sestavenému rozvrhu asi v řádu desítek procent nemalý úspěch.

Podíváme-li se na průměrné hodnoty, zjistíme, že nejlepších výsledků dosahoval Genetický algoritmus při populaci 1000 jedinců a 800 iterací. Zde to bylo průměrně až 21 % nárůst kvality oproti ručně sestavenému plánu.

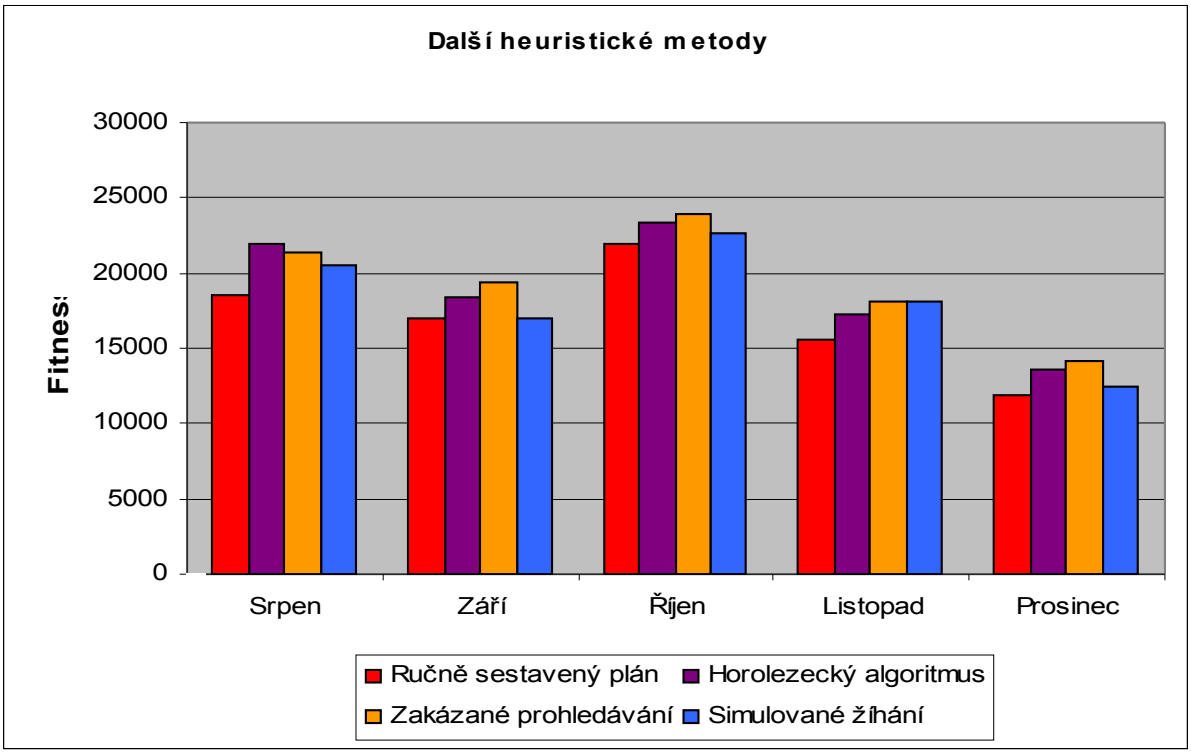


Obr. 25 Srovnání fitness pro Genetické algoritmy.

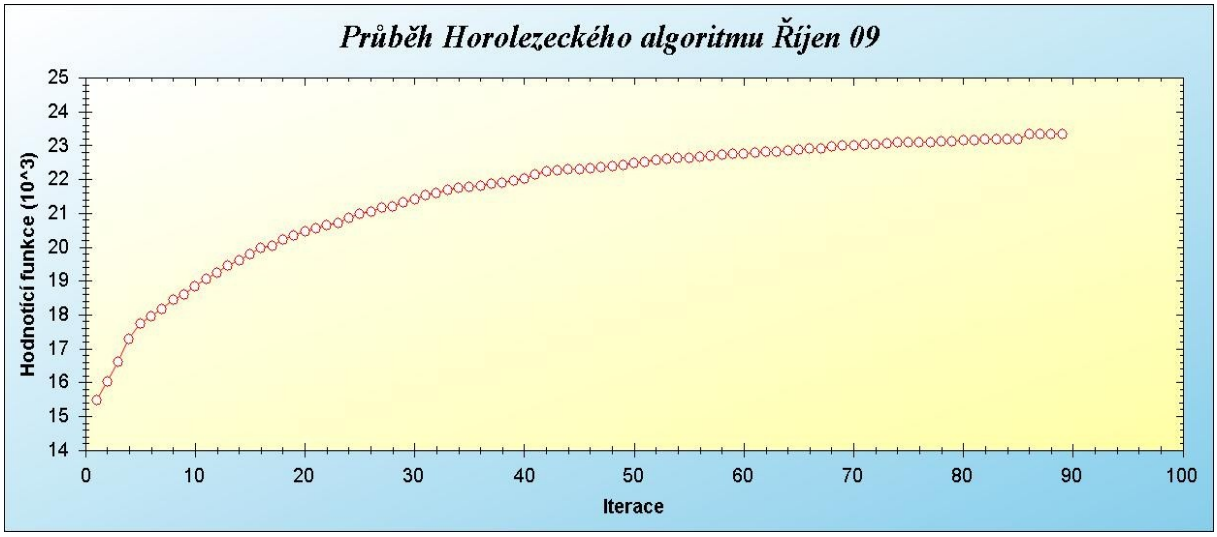


Obr. 26 Srovnání průměrných hodnot fitness.

Podíváme-li se na Horolezecký algoritmus, zjistíme, že tento algoritmus opravdu skončil u každého měsíce předčasně, tj. uvízl v lokálním minimu. Průměrně dosahoval nárůstu kvality o 12 % a zastavil se na 83 iterací při prohledávaném okolí v každé iteraci nastavené na 1000 plánů. To je výsledek, který je srovnatelný se základním nastavením u Genetických algoritmů, který dosahoval u 100 iterací a populaci 400 jedinců nárůstu o 9 %. U Horolezeckého algoritmu by nárůst iterací nepomohl k nalezení lepšímu řešení. Z *Obr. 28* je opravdu patrné, že se jedná o gradientní metodu, kde fitness funkce slouží jako ukazatel největšího spádu a křivka průběhu fitness má nádherný tvar s rostoucí tendencí.

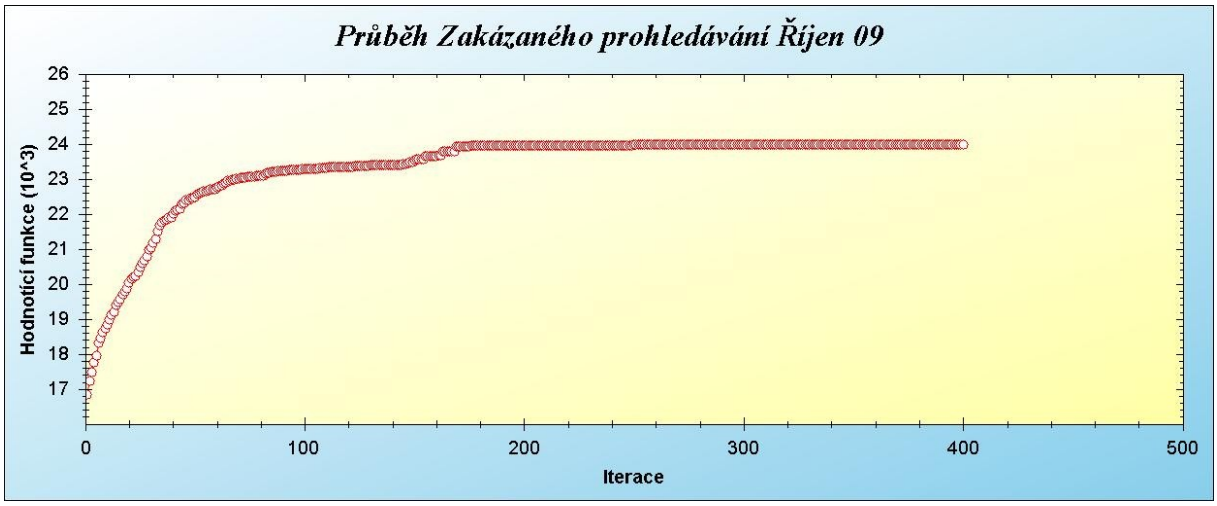


Obr. 27 Srovnání fitness pro ostatní heuristické metody.



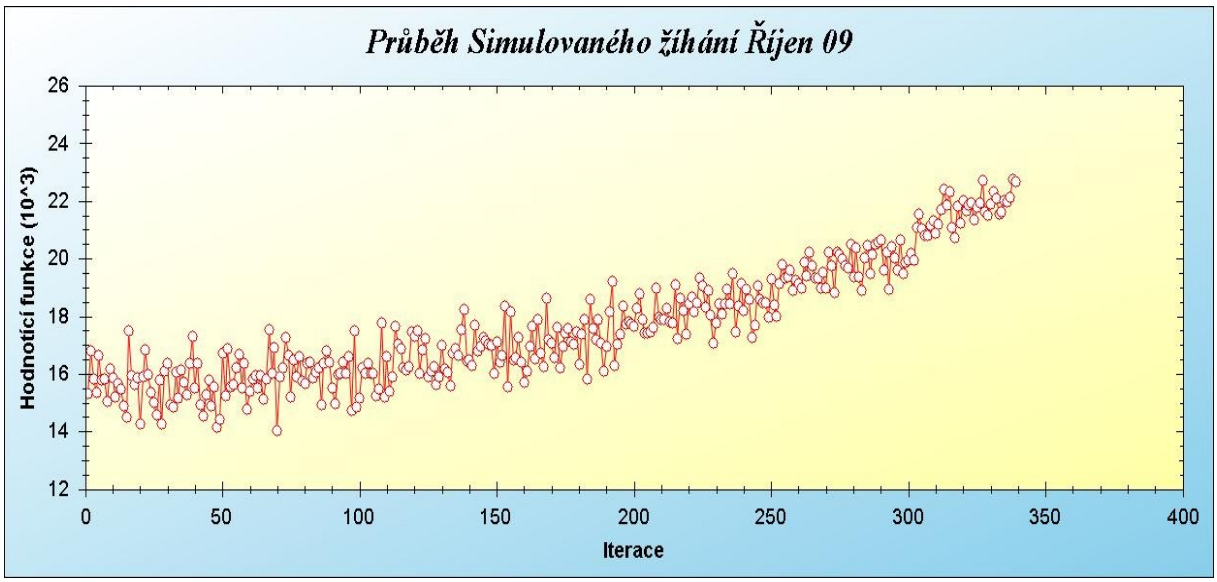
Obr. 28 Průběh fitness funkce Horolezeckého algoritmu.

Vylepšená verze Horolezeckého algoritmu Zakázané prohledávání měla z použitých základních heuristik nejlepší výsledky. Bylo zde použito 400 iterací, prohledávané okolí v každé iteraci bylo 1000 plánů a zakázaný seznam obsahoval 25 plánů. Výsledný průměrný nárůst kvality řešení je o 15 %, což je srovnatelný výsledek s druhým nastavením Genetického algoritmu na populaci 1000 jedinců a 400 iterací, kde se průměrně dosahovalo nárůstu kvality o 16 %.



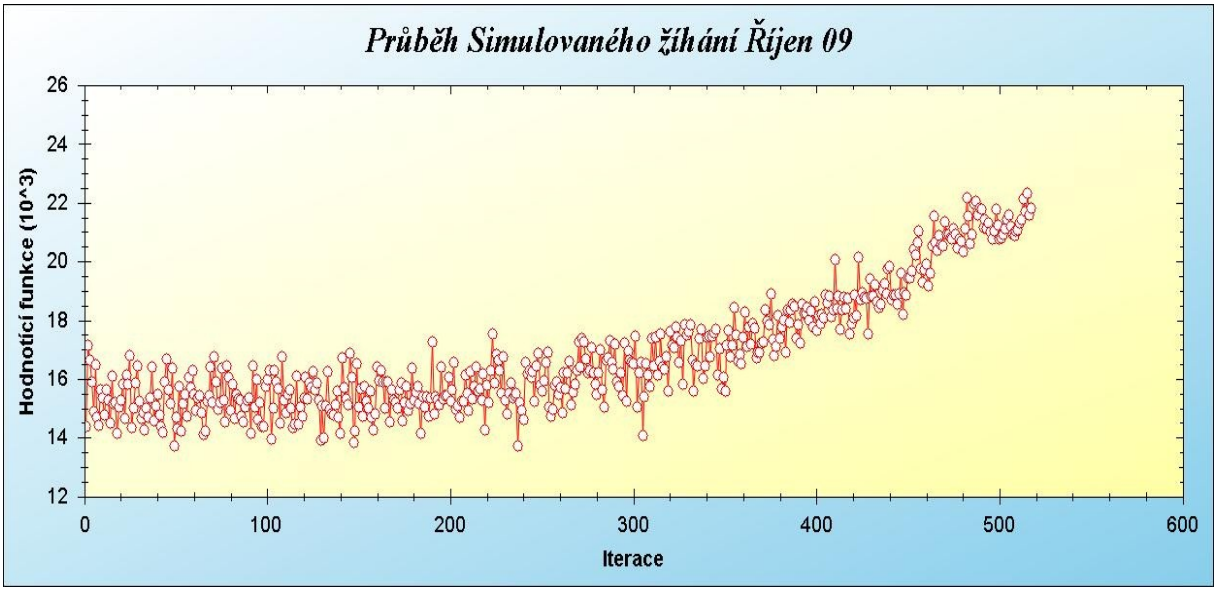
Obr. 29 Průběh fitness funkce Zakázané prohledávání.

Nicméně Zakázané prohledávání se zde dostalo téměř nad své možnosti, ještě je nějaká šance, že při jiném nastavení okolí a zakázaného seznamu by algoritmus mohl dosáhnout lepšího výsledku, ale jak je patrné z *Obr. 29*, průběh fitness funkce po určité iteraci začne stagnovat a algoritmu se nedaří najít lepší řešení, a proto uvízne v lokálním minimu. Je to dáno konstrukcí algoritmu. Generování okolí probíhá tak, že nový plán je vygenerován na základě přehození dvou úkolů na jiná místa. Taková modifikace nestačí k tomu, aby se algoritmus vymanil z lokálního minima a byl schopen pokračovat v jiném hlubším lokálním minimu, které se nachází při mnohem podstatnější změně. To může být výhodou u Genetického algoritmu, kde je určitá pravděpodobnost podstatné změny rozvržení fenotypu v konkrétní iteraci.



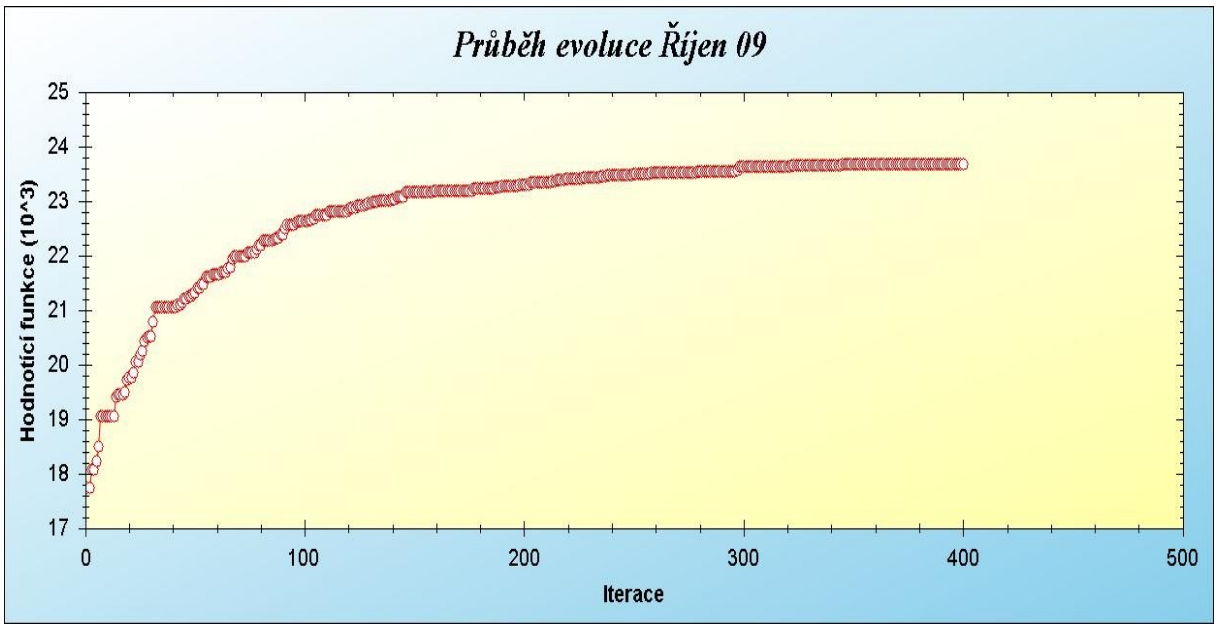
Obr. 30 Průběh fitness funkce Simulovaného žíhání.

Jak je patrné simulované žíhání *Obr. 30,* za své aktuální řešení bere i lokálně horší řešení, což může být výhodou při prohledávání. V našem případě bylo nastaveno alfa na hodnotu 0,99, teplota žíhání v rozmezí 50 – 1500 a parametr k_{max} na hodnotu 1000. Doba výpočtu byla kolem 20 minut, přesto se algoritmu i při jiném nastavení nedařilo dosáhnout lepších výsledků než je 9 % zlepšení kvality oproti ručně sestavenému plánu.



Obr. 31 Průběh fitness funkce Simulovaného žíhání pro vyšší teplotu.

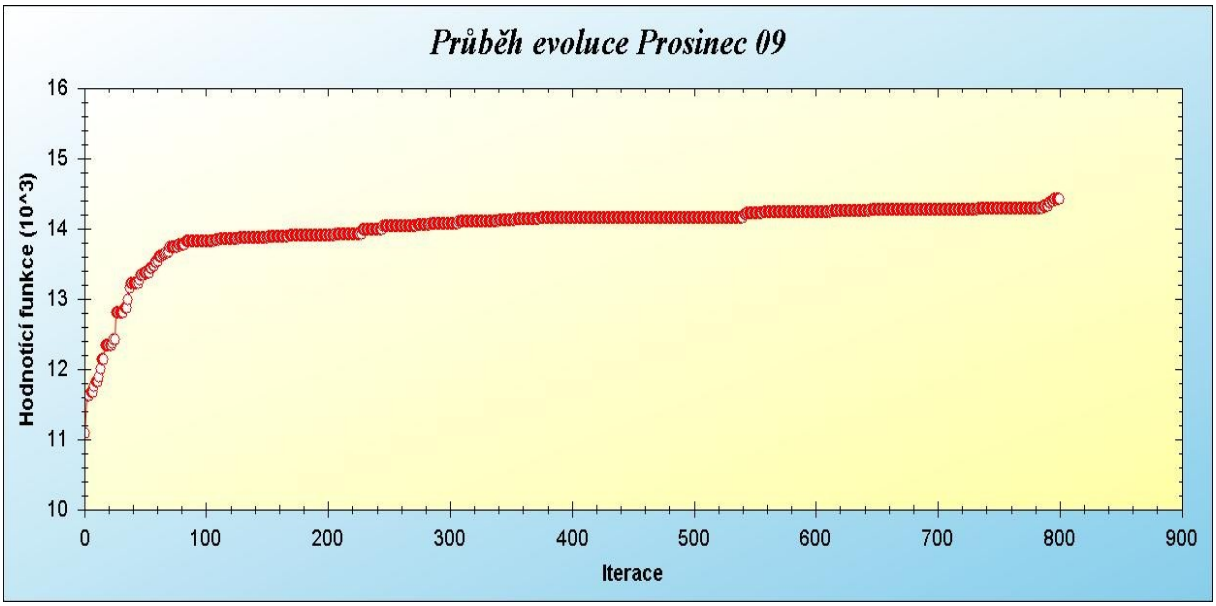
Otázkou je, kde jsou meze tohoto algoritmu pro tuto konkrétní praktickou aplikaci, zda by nebylo možné docílit lepších výsledků při vyšších nebo nižších žíhacích teplotách. Pro tento účel, jak je možné vidět na *Obr. 31*, bylo spuštěno hledání optima s nastavenou žíhací teplotou v rozmezí 50 až 9000. Na základě této zkoušky bylo zjištěno, že ani pří více iteracích algoritmus nedosahuje lepších výsledků. Hodnota fitness funkce byla 21833, to je dokonce horší výsledek než pro ručně sestavený plán a doba výpočtu byla 43 minut.



Obr. 32 Průběh fitness funkce Genetického algoritmu pro plán Říjen 09.

Jak je vidět, u Simulovaného žíhání záleží na žíhací teplotě, protože na základě ní je díky Metropolisova kritéria (kapitola 5.) při vyšší teplotě akceptováno velké množství nově vzniklých stavů, v našem případě plánů. Při nižších teplotách je akceptováno zase málo nově vzniklých stavů a algoritmus není opět schopen najít řešení lepší než v řádu 9 % oproti ručně sestavenému řešení.

Z Obr. 32 a 33 je patrné, že u Genetického algoritmu dochází k nepatrnému vylepšování v celém průběhu výpočtu, a tím je možné na základě delšího výpočtu dosáhnout lepších řešení. Jak je vidět z názorných tabulek, Genetické algoritmy jako robustnější heuristická metoda dosahovala lepších výsledků než uvedené základní heuristické metody. To je dáno schopností Genetického algoritmu překonat lokální minima na základě více stochastických prvků, které jsou součástí myšlenky genetické evoluce.



Obr. 33 Průběh fitness funkce Genetického algoritmu pro plán Prosinec 09.

9 ZÁVĚR

Práce se zabývala konkrétním problémem rozvrhování výroby ve výrobním družstvu Obzor Zlín, stejně jako teoretickým aspektům této problematiky. Na základě této práce bude možné sestavovat rozvrhy pomocí počítače v řádu minut, a ušetřit tak několikadenní práci inženýra. Z praktické části vyplývá, že Genetické algoritmy dosahovaly největšího nárůstu kvality oproti ručně sestaveným plánům při 800 iteracích a velikosti populace 1000 jedinců. Nárůst kvality takto sestavovaných rozvrhů byl průměrně o 21 %. Další heuristické metody, které byly porovnávány s Genetickými algoritmy, dosahovaly nárůstu kvality jen v řádu 10 % až 15 %. Tyto jejich výsledky byly tedy srovnatelné s Genetickým algoritmem při nastavení populace na 400 jedinců a počtu iterací 100.

K výsledkům je potřeba podotknout, že Genetické algoritmy by mohly dosahovat lepších výsledků při jejich použití na více průchodů anebo při postupném sestavování plánů. Například by bylo možné v různých průchodech optimalizovat jen na určité aspekty fitness funkce a tímto získat předběžné plány, které budou následně použity jako výchozí jedinci pro konečnou optimalizaci.

Do budoucna by nebylo špatné srovnat tyto výsledky s dalšími heuristickými přístupy anebo jinými metodami a také zahrnout možnost dynamické změny již rozplánované výroby. V provozu se stává, že některé zakázky přicházejí v průběhu měsíce, takže by bylo dobré mít možnost je přidat do stávajícího plánu a udělat nový plán s ohledem na rozpracovanost aktuální výroby.

SEZNAM POUŽITÉ LITERATURY

[1] OBZOR, výrobní družstvo Zlín. *Profil* [online]., 2007. [cit. 2010-04-08].
Dostupné z: http://www.obzor.cz/cz/kategorie/profil.aspx

[2] Molde university college. *Log 904 Seminars in Logistics*, Scheduling Models and Algorithms[online]., [cit. 2010-04-09].
Dostupné z: http://himolde.studiehandbok.no/eng/content/view/full/11001/language/eng-GB

[3] University of Osnabrück, Germany. *Complexity results for scheduling problems* [online]., 29-06-2009. [cit. 2010-04-09].
Dostupné z: http://www.mathematik.uni-osnabrueck.de/research/OR/class/

[4] Fakulta Informatiky MU Brno. *Úvod do rozvrhování* [online]., 23-02-2010. [cit. 2010-04-09].
Dostupné z: http://www.fi.muni.cz/~hanka/rozvrhovani/prusvitky/prvni_bw.pdf

[5] Cornell University. *Minimizing the Makespan for Unrelated Parallel Machines* [online]., 06-01-2003. [cit. 2010-04-09].
Dostupné z: http://www.cs.cornell.edu/~guoys/publications/MachineScheduleYL.pdf

[6] Bílek, Ivo. Aproximativní a heuristické metody řešení NP-těžkých problémů.VUT v Brně FSI. 2007. 46 s.

[7] Wikipedie. *Heuristika* [online]., 11-04-2010. [cit. 2010-04-09].
Dostupné z: http://cs.wikipedia.org/wiki/Heuristika
Z. Michalewicz and D.B. Fogel, How to Solve It: Modern Heuristics, Springer,. 2000.

[8] Applegate, D. L.; Bixby, R. M.; Chvátal, V.; Cook, W. J. (2006), The Traveling Salesman Problem, ISBN 0691129932.

[9] Zdeněk Vašíček. Simulované žíhání. VUT v Brně FIT. 9 s. [cit. 2010-04-09].
Dostupné z: http://www.stud.fit.vutbr.cz/~xvasic11/projects/msi.pdf

[10] Hynek, Josef: Genetické algoritmy a genetické programování. Grada Publishing, 2008, 200 s. (ISBN: 978 80 247 2695 3).

[11] Prata, Stephen: *Mistrovství v C++, 3. aktualizované vydání*. Computer Press, 10/2008, ISBN: 978 80 251 1749 1, EAN: 978 80 251 1749 1

[12] Paolo Pialorsi, Marco Russo. *Microsoft LINQ Kompletní průvodce programátora.* Computer Press, 9/2009, ISBN: 978-80-251-2735-3 EAN: 9788025127353

[13] Kolektiv autorů (Simon Robinson, K. Scott Allen, Ollie Cornes, Jay Glynn, Zach Greenvoss, Burton Harvey, Christian Nagel, Morgna Skinner, Karli Watson). *C# Programujeme profesionálně.* První vydání. Brno : Computer Press, 2003. 1130 s. ISBN 80-251-0085-5

[14] Tomáš J. Kouba. Test rychlosti: Java, C# a C++ na MS Windows a Java na Linuxu [online]., 09-05-2006. [cit. 2010-04-09].
Dostupné z: http://tomas-net.blogspot.com/2006/05/test-rychlosti-java-c-c-na-ms-windows.html

[15]Wikipedie. Minitab [online]., 28-03-2010. [cit. 2010-04-09].
Dostupné z: http://cs.wikipedia.org/wiki/Minitab

[16]Donald E. Knuth: The Art of Computer Programming, Volume 2: Seminumerical Algorithms, (Addison-Wesley, 1997), ISBN 0-201-89684-2.

[17]Jakub Galgonek, Michal Heppler, Petr Janata, Alexandr Kazda, Tomáš Urban. Generátory pseudonáhodných čísel [online]. [cit. 2010-04-09].
Dostupné z: http://fyztyd.fjfi.cvut.cz/2001/web/sbornik/pdf/nah.pdf

[18]Extreme Optimization. Random Number Generators [online].2003. [cit. 2010-04-09]. Dostupné z: http://www.extremeoptimization.com/Documentation/Statistics/Random_Numbers/Random_Number_Generators.aspx

[19]Wikipedie. Metoda Monte Carlo [online]., 24-04-2010. [cit. 2010-04-28].
Dostupné z: http://cs.wikipedia.org/wiki/Metoda_Monte_Carlo

[20]Wikipedie. Cesàro summation [online]., 06-02-2010. [cit. 2010-04-28].
Dostupné z: http://en.wikipedia.org/wiki/Ces%C3%A0ro_summation

[21]Mersenne Twister: A Study on Random Number Generators [online]. [cit. 2010-04-09].
Dostupné z: http://student.vub.ac.be/~nkaraogl/mt/mt.html

[22]Pohlheim, Hartmut: GEATbx - The Genetic and Evolutionary Algorithm Toolbox for Matlab [Online Documentation]. [cit. 2010-04-09].
Dostupné z: http://www.geatbx.com/

[23]Kajánek, Petr; Přikryl, Josef: Seznámení se s genetickými algoritmy. [cit. 2010-04-09].
Dostupné z: http://mujweb.cz/www/prikrylj/Genetickealgoritmy.html#6.4